Linux Amateur Radio AX.25 HOWTO

Jeff Tranter, VE3ICH

tranter@pobox.com

v2.1, 19 September 2001

The Linux operating system is perhaps the only operating system in the world that can boast native and standard support for the AX.25 packet radio protocol utilized by Amateur Radio operators worldwide. This document describes how to install and configure this support.

1. Introduction

Amateur radio is a non-profit, non-commercial activity enjoyed by hobbyists world-wide. Radio amateurs are licensed by government authorities to use portions of the radio spectrum allocated to them for non-commercial, non-profit activities including personal communication, public service, and technical experimentation. Packet Radio is a particular digital mode of communication that makes use of networking protocols to provide computer to computer communication.

This document was originally an appendix to the HAM-HOWTO, but grew too large to be reasonably managed in that fashion. This document describes how to install and configure the native AX.25, NET/ROM and ROSE support for Linux. A few typical configurations are described that could be used as models to work from.

The Linux implementation of the amateur radio protocols is very flexible. To people relatively unfamiliar with the Linux operating system the configuration process may look daunting and complicated. It will take you a little time to come to understand how the whole thing fits together. You will find configuration very difficult if you have not properly prepared yourself by learning about Linux in general. You cannot expect to switch from some other environment to Linux without learning about Linux itself.

1.1. Changes from the previous version

- Updated IPIP tunnelling section to reflect iproute2 package (thanks to Milan Kalina).

1.2. Where to obtain new versions of this document

The best place to obtain the latest version of this document is from a Linux Documentation Project archive. The Linux Documentation Project runs a web server and this document appears there as the AX25-HOWTO (http://www.linuxdoc.org/HOWTO/AX25-HOWTO.html). This document is also available in various formats from the Linux Documentation Project (http://www.linuxdoc.org).

You can always contact me, but I pass new versions of the document directly to the LDP HOWTO coordinator, so if it isn't there then chances are I haven't finished it.

1.3. Other related documentation

There is a lot of related documentation. There are many documents that relate to Linux networking in more general ways and I strongly recommend you also read these as they will assist you in your efforts and provide you with deeper insight into other possible configurations. They are:

- Linux Networking HOWTO (http://www.linuxdoc.org/HOWTO/Net-HOWTO/index.html)
- Linux Ethernet HOWTO (http://www.linuxdoc.org/HOWTO/Ethernet-HOWTO.html)
- Linux Firewall and Proxy Server HOWTO (http://www.linuxdoc.org/HOWTO/Firewall-HOWTO.html)
- Linux 2.4 Advanced Routing HOWTO (http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html)
- Netrom-Node mini-Howto (http://www.linuxdoc.org/HOWTO/mini/Netrom-Node.html)

You may come across references to a Linux HAM HOWTO. This document is obsolete and has been replaced by the Hamsoft Linux Ham Radio Applications and Utilities Database (http://radio.linux.org.au/) web site. More general Linux information may be found by referencing other Linux HOWTO (http://www.linuxdoc.org/HOWTO/HOWTO-INDEX/index.html) documents.

2. The Packet Radio Protocols and Linux

The *AX.25* protocol offers both connected and connectionless modes of operation, and is used either by itself for point-point links, or to carry other protocols such as TCP/IP and NET/ROM.

It is similar to X.25 level 2 in structure, with some extensions to make it more useful in the amateur radio environment.

The NET/ROM protocol is an attempt at a full network protocol and uses AX.25 at its lowest layer as a datalink protocol. It provides a network layer that is an adapted form of AX.25. The NET/ROM protocol

features dynamic routing and node aliases.

The ROSE protocol was conceived and first implemented by Tom Moulton W2VY and is an implementation of the X.25 packet layer protocol and is designed to operate with AX.25 as its datalink layer protocol. It too provides a network layer. ROSE addresses take the form of 10 digit numbers. The first four digits are called the Data Network Identification Code (DNIC) and are taken from Appendix B of the CCITT X.121 recommendation. More information on the ROSE protocol may be obtained from the RATS Web server (http://www.rats.org/).

Alan Cox developed some early kernel based AX.25 software support for Linux. Jonathon Naylor (mailto:g4klx@g4klx.demon.co.uk) has taken up ongoing development of the code, has added NET/ROM and ROSE support and is now the developer of the AX.25 related kernel code. DAMA support was developed by Joerg (mailto:jreuter@poboxes.com), DL1BKE. Baycom and Soundmodem support were added by Thomas Sailer (mailto:sailer@ife.ee.ethz.ch). The AX.25 software is now maintained by a small team of developers on SourceForge (http://www.sourceforge.net).

The Linux code supports KISS and 6PACK based TNC's (Terminal Node Controllers), the Ottawa PI card, the Gracilis PacketTwin card and other Z8530 SCC based cards with the generic SCC driver, several parallel and serial port Baycom modems, and serial port YAM modems. Thomas Sailer's kernel soundmodem driver supports SoundBlaster and sound cards based on the Crystal chip set, and his newer user-mode soundmodem uses the standard kernel sound drivers, so it should work with any sound card supported under Linux.

The user programs contain a simple PMS (Personal Message System), a beacon facility, a line mode connect program, `listen` (an example of how to capture all AX.25 frames at raw interface level), and programs to configure the NET/ROM protocol. Also included are an AX.25 server style program to handle and dispatch incoming AX.25 connections and a NET/ROM daemon which does most of the hard work for NET/ROM support.

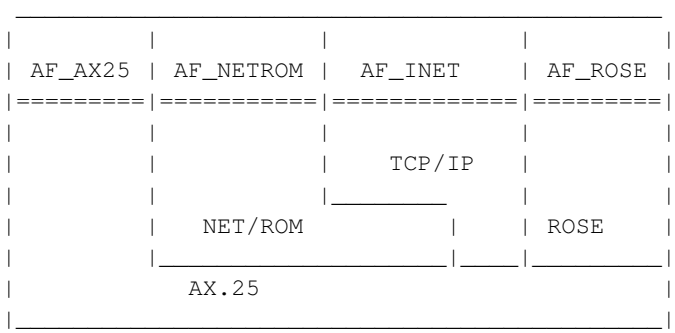
There are utility programs to support APRS, including digipeating and gatewaying to the Internet.

2.1. How it all fits together

The Linux AX.25 implementation is a brand new implementation. While in many ways it may looks similar to NOS, or BPQ or other AX.25 implementations, it is none of these and is not identical to any of them. The Linux AX.25 implementation is capable of being configured to behave almost identically to other implementations, but the configuration process is very different.

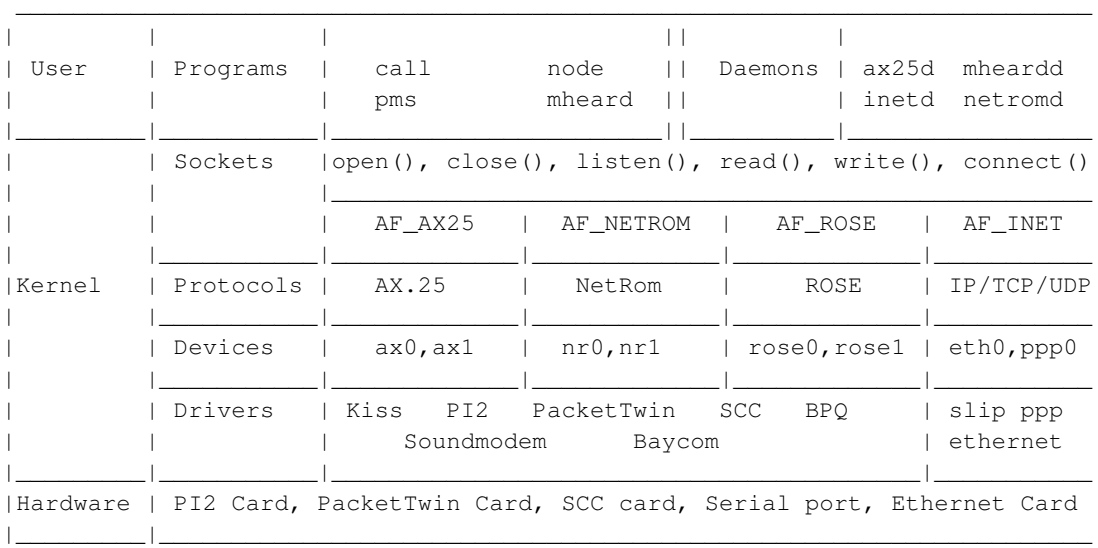
To assist you in understanding how you need to think when configuring this section describes some of the structural features of the AX.25 implementation and how it fits into the context of the overall Linux structure.

Simplified Protocol Layering Diagram



This diagram simply illustrates that NET/ROM, ROSE and TCP/IP all run directly on top of AX.25, but that each of these protocols is treated as a separate protocol at the programming interface. The '_' names are simply the names given to the '*Address Family*' of each of these protocols when writing programs to use them. The important thing to note here is the implicit dependence on the configuration of your AX.25 devices before you can configure your NET/ROM, ROSE or TCP/IP devices.

Software Module Diagram of Linux Network Implementation



This diagram is a little more general than the first. This diagram attempts to show the relationship between user applications, the kernel and the hardware. It also shows the relationship between the Socket application programming interface, the actual protocol modules, the kernel networking devices and the device drivers. Anything in this diagram is dependent on anything underneath it, and in general you must configure from the bottom of the diagram upwards. So for example, if you want to run the *call* program

you must also configure the hardware, then ensure that the kernel has the appropriate device driver, that you create the appropriate network device, that the kernel includes the desired protocol that presents a programming interface that the *call* program can use. I have attempted to lay out this document in roughly that order.

3. The AX.25/NET/ROM/ROSE software components

The AX.25 software is comprised of three components: the kernel source, the network configuration tools and the utility programs.

AX.25 support in the Linux kernel has been fairly stable since the 2.2 series of kernel versions. This document assumes you are using the most recent kernel, which as the time of writing was 2.4.9.

> **Note:** Software versions listed in this document were the latest at the time of writing, but are subject to change. Check for newer versions when downloading them.

3.1. Finding the kernel, tools and utility packages

3.1.1. The kernel source

The kernel source can be found at `www.kernel.org` and `ftp.kernel.org`. For the 2.4.9 kernel it would be downloaded from ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.9.tar.gz.

3.1.2. The network tools

The latest release of the standard Linux network tools support AX.25 and NET/ROM and can be found at http://www.tazenda.demon.co.uk/phil/net-tools.

The latest ipchains package can be found at http://netfilter.filewatcher.org/ipchains (http://netfilter.filewatcher.org/ipchains/).

> **Note:** It is usually not necessary to download and install these as any recent Linux distribution should include them.

3.1.3. The AX.25 utilities

The old ax25-utils used with the 2.0 and 2.1 kernels is now obsolete and has been replaced with new packages hosted on SourceForge (http://sourceforge.net) at http://sourceforge.net/projects/hams.

The software is distributed as three packages: the AX.25 library, tools, and applications. At the time of writing the most recent versions were the following:

- ftp://hams.sourceforge.net/pub/hams/ax25/libax25-0.0.7.tar.gz
- ftp://hams.sourceforge.net/pub/hams/ax25/ax25-tools-0.0.6.tar.gz
- ftp://hams.sourceforge.net/pub/hams/ax25/ax25-apps-0.0.4.tar.gz

3.1.4. The APRS utilities

If you want to use APRS you can download aprsd (http://sourceforge.net/projects/aprsd/) and aprsdigi (http://www.users.cloud9.net/~alan/ham/aprs/):

- http://prdownloads.sourceforge.net/aprsd/aprsd-2.1.4.tar.gz
- http://www.users.cloud9.net/~alan/ham/aprs/aprsdigi-2.0-pre3.tar.gz

4. Installing the AX.25/NET/ROM/ROSE software

To successfully install AX.25 support on your Linux system you must configure and install an appropriate kernel and then install the AX.25 utilities.

> **Tip:** Rather than building and installing from source, you may prefer to install prebuilt binary packages for your system. Debian and RPM format packages are available on various archive sites including http://www.debian.org and http://rpmfind.net; look for "ax25". Incidently, the Debian Linux distribution is considered by many people to be one of the more "Amateur Radio friendly" distributions, and provides many amateur radio applications as Debian packages (one of the founders of the project is a ham).

4.1. Compiling the kernel

If you are already familiar with the process of compiling the Linux kernel then you can skip this section, just be sure to select the appropriate options when compiling the kernel. If you are not, then read on. You

may also want to read the Linux Kernel HOWTO (http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html).

The normal place for the kernel source to be unpacked to is the `/usr/src` directory into a subdirectory called `linux`. To do this you should be logged in as `root` and execute a series of commands similar to the following:

```
# cd /usr/src
# mv linux linux.old
# tar xzvf linux-2.4.9.tar.gz
# cd linux
```

After you have unpacked the kernel source, you need to run the configuration script and choose the options that suit your hardware configuration and the options that you wish built into your kernel. You do this by using the command:

```
# make menuconfig
```

If you are running X you can get a graphical interface using:

```
# make xconfig
```

You might also try:

```
# make config
```

I'm going to describe the full screen method (menuconfig) because it is easier to move around, but use whichever you are most comfortable with.

In either case you will be offered a range of options at which you must answer 'Y' or 'N'. (Note you may also answer 'M' if you are using modules. For the sake of simplicity I will assume you are not, please make appropriate modifications if you are).

The options most relevant to an AX.25 configuration are:

```
Code maturity level options  --->
    [*] Prompt for development and/or incomplete code/drivers
    ...
General setup  --->
    ...
    [*] Networking support
    ...
Networking options  --->
    <*> UNIX domain sockets
    ...
    [*] TCP/IP networking
    ...
    [?] IP: tunneling
    ...
Amateur Radio Support --->
    --- Packet Radio protocols
    [*]   Amateur Radio AX.25 Level 2 protocol
    [?]     AX.25 DAMA Slave support
    [?]     Amateur Radio NET/ROM protocol
    [?]     Amateur Radio X.25 PLP (Rose)
    AX.25 network device drivers  --->
    <?> Serial port KISS driver
    <?> Serial port 6PACK driver
    <?> BPQ Ethernet driver
    <?> High-speed (DMA) SCC driver for AX.25
    <?> Z8530 SCC driver
    <?> BAYCOM ser12 fullduplex driver for AX.25
    <?> BAYCOM ser12 halfduplex driver for AX.25
    <?> BAYCOM picpar and par96 driver for AX.25
    <?> BAYCOM epp driver for AX.25
    <?> Soundcard modem driver
    [?]   soundmodem support for Soundblaster and compatible cards
    [?]   soundmodem support for WSS and Crystal cards
    [?]   soundmodem support for 1200 baud AFSK modulation
    [?]   soundmodem support for 2400 baud AFSK modulation (7.3728MHz crystal)
    [?]   soundmodem support for 2400 baud AFSK modulation (8MHz crystal)
    [?]   soundmodem support for 2666 baud AFSK modulation
    [?]   soundmodem support for 4800 baud HAPN-1 modulation
    [?]   soundmodem support for 4800 baud PSK modulation
    [?]   soundmodem support for 9600 baud FSK G3RUH modulation
    <?> YAM driver for AX.25
```

The options I have flagged with a '*' are those that you *must* must answer 'Y' to. The rest are dependent on what hardware you have and what other options you want to include. Some of these options are described in more detail later on, so if you don't know what you want yet, then read ahead and come back to this step later.

After you have completed the kernel configuration you should be able to cleanly compile your new kernel:

```
# make dep
# make clean
# make zImage
```

Make sure you move your `arch/i386/boot/zImage` file wherever you want it and then edit your `/etc/lilo.conf` file and rerun *lilo* to ensure that you actually boot from it.

4.1.1. A word about kernel modules

Compiling drivers as modules is useful if you only use AX.25 occasionally and want to be able to load and unload them on demand to save system resources. However, some people have problems getting the modularized drivers working because they are more complicated to configure. If you've chosen to compile any drivers as modules, then you'll also need to run the commands:

```
# make modules
# make modules_install
```

to install your modules in the appropriate location.

You will also need to add some entries into your `/etc/modules.conf` file to ensure that the *kerneld* program knows how to locate the kernel modules. You should add/modify the following:

```
alias net-pf-3     ax25
alias net-pf-6     netrom
alias net-pf-11    rose
alias tty-ldisc-1  slip
alias tty-ldisc-3  ppp
alias tty-ldisc-5  mkiss
alias bc0          baycom
alias nr0          netrom
alias pi0a         pi2
alias pt0a         pt
alias scc0         optoscc    (or one of the other scc drivers)
alias sm0          soundmodem
alias tunl0        newtunnel
alias char-major-4 serial
```

```
alias char-major-5 serial
alias char-major-6 lp
```

> **Tip:** On Debian-based Linux systems these entries should go into the file `/etc/modutils/aliases` and then you need to run `/sbin/update-mpodules`.

4.2. The AX.25 library, tools, and application programs

After you have successfully compiled and booted your new kernel you need to compile and install the ax25 library, tools, and application programs.

To compile and install libax25 you should use a series of commands similar to the following:

```
# cd /usr/src
# tar xzvf libax25-0.0.7.tar.gz
# cd libax25-0.0.7
# ./configure --exec_prefix=/usr --sysconfdir=/etc --localstatedir=/var
# make
# make install
```

> **Tip:** The arguments to the `configure` command ensure that the files will be installed in the "standard" places under the directory `/usr` in subdirectories `bin`, `sbin`, `etc` and `man`. If you simply run configure with no options it will default to putting all files under `/usr/local`. This can cause the situation where you have configuration files in both `/usr` and `/usr/local`. If you want to ensure that this can't happen you can make `/usr/local/etc/ax25` a symbolic link to `/etc/ax25` at the very beginning of the install process and then you won't have to worry about it.

If this is a first time installation, that is you've never installed any ax25 code on your machine before, you should also use the:

```
# make installconf
```

command to install some sample configuration files into the `/etc/ax25/` directory from which to work.

You can now build install the AX.25 tools in a similar fashion:

```
# cd /usr/src
# tar xzvf ax25-tools-0.0.6.tar.gz
# cd ax25-tools-0.0.6
# ./configure --prefix=/usr  --sysconfdir=/etc --localstatedir=/var
# make
# make install
# make installconf (if you want to install the configuration files)
```

And finally you can install the AX.25 applications:

```
# cd /usr/src
# tar xzvf ax25-apps-0.0.4.tar.gz
# cd ax25-apps-0.0.4
# ./configure --prefix=/usr  --sysconfdir=/etc --localstatedir=/var
# make
# make install
# make installconf (if you want to install the configuration files)
```

If you get messages something like:

```
gcc -Wall -Wstrict-prototypes -O2 -I../lib -c call.c
call.c: In function 'statline':
call.c:268: warning: implicit declaration of function 'attron'
call.c:268: 'A_REVERSE' undeclared (first use this function)
call.c:268: (Each undeclared identifier is reported only once
call.c:268: for each function it appears in.)
```

then you should double check that you have the *ncurses* package properly installed on your system. The configuration script attempts to locate your package in the common locations, but some installations have it badly installed and it is unable to locate them.

5. A note on callsigns, addresses and things before we start

Each AX.25 and NET/ROM port on your system must have a callsign/ssid allocated to it. These are configured in the configuration files that will be described in detail later on.

Some AX.25 implementations such as NOS and BPQ will allow you to configure the same callsign/ssid on each AX.25 and NET/ROM port. For somewhat complicated technical reasons Linux does not allow this. This isn't as big a problem in practice as it might seem.

This means that there are things you should be aware of and take into consideration when doing your configurations.

1. Each AX.25 and NET/ROM port must be configured with a unique callsign/ssid.
2. TCP/IP will use the callsign/ssid of the AX.25 port it is being transmitted or received by, ie the one you configured for the AX.25 interface in point 1.
3. NET/ROM will use the callsign/ssid specified for it in its configuration file, but this callsign is only used when your NET/ROM is speaking to another NET/ROM, this is *not* the callsign/ssid that AX.25 users who wish to use your NET/ROM 'node' will use. More on this later.
4. ROSE will, by default, use the callsign/ssid of the AX.25 port, unless the ROSE callsign has been specifically set using the '*rsparms*' command. If you set a callsign/ssid using the '*rsparms*' command then ROSE will use this callsign/ssid on all ports.
5. Other programs, such as the '*ax25d*' program can listen using any callsign/ssid that they wish and these may be duplicated across different ports.
6. If you are careful with routing you can configure the same IP address on all ports if you wish.

5.1. What are all those T1, T2, N2 and things ?

Not every AX.25 implementation is a TNC2. Linux uses nomenclature that differs in some respects from that you will be used to if your sole experience with packet is a TNC. The following table should help you interpret what each of the configurable items are, so that when you come across them later in this text you'll understand what they mean.

Linux	TAPR TNC	Description
T1	FRACK	How long to wait before retransmitting an unacknowledged frame.

Linux	TAPR TNC	Description
T2	RESPTIME	The minimum amount of time to wait for another frame to be received before transmitting an acknowledgement.
T3	CHECK	The period of time we wait between sending a check that the link is still active.
N2	RETRY	How many times to retransmit a frame before assuming the connection has failed.
Idle		The period of time a connection can be idle before we close it down.
Window	MAXFRAME	The maximum number of unacknowledged transmitted frames.

5.2. Run time configurable parameters

The kernel allows you to change many parameters at run time. If you take a careful look at the `/proc/sys/net/` directory structure you will see many files with useful names that describe various parameters for the network configuration. The files in the `/proc/sys/net/ax25/` directory each represent one configured AX.25 port. The name of the file relates to the name of the port.

The structure of the files in `/proc/sys/net/ax25/`*`portname`*`/` is as follows:

Filename	Meaning	Values	Default
ip_default_mode	IP Default Mode	0=DG 1=VC	0
ax25_default_mode	AX.25 Default Mode	0=Normal 1=Extended	0
backoff_type	Backoff	0=Linear 1=Exponential	1
connect_mode	Connected Mode	0=No 1=Yes	1
standard_window_size	Standard Window	1 .. 7	2
extended_window_size	Extended Window	1 .. 63	32
t1_timeout	T1 Timeout	1s .. 30s	10s
t2_timeout	T2 Timeout	1s .. 20s	3s
t3_timeout	T3 Timeout	0s .. 3600s	300s
idle_timeout	Idle Timeout	0m or greater	20m

Filename	Meaning	Values	Default
maximum_retry_count	N2	1 .. 31	10
maximum_packet_length	AX.25 Frame Length	1 .. 512	256

In the table T1, T2 and T3 are given in seconds, and the Idle Timeout is given in minutes. But please note that the values used in the sysctl interface are given in internal units where the time in seconds is multiplied by 10, this allows resolution down to 1/10 of a second. With timers that are allowed to be zero, e.g. T3 and Idle, a zero value indicates that the timer is disabled.

The structure of the files in `/proc/sys/net/netrom/` is as follows:

Filename	Meaning	Values
default_path_quality		
link_fails_count		
network_ttl_initialiser		
obsolescence_count_initialiser		
routing_control		
transport_acknowledge_delay		
transport_busy_delay		
transport_maximum_tries		
transport_requested_window_size		
transport_timeout		

The structure of the files in `/proc/sys/net/rose/` is as follows:

Filename	Meaning	Values
acknowledge_hold_back_timeout		
call_request_timeout		
clear_request_timeout		
link_fail_timeout		

Filename	Meaning	Values
maximum_virtual_circuits		
reset_request_timeout		
restart_request_timeout		
routing_control		
window_size		

To set a parameter all you need to do is write the desired value to the file itself, for example to check and set the ROSE window size you'd use something like:

```
# cat /proc/sys/net/rose/window_size
3
# echo 4 >/proc/sys/net/rose/window_size
# cat /proc/sys/net/rose/window_size
4
```

6. Configuring an AX.25 port

Each of the AX.25 applications read a particular configuration file to obtain the parameters for the various AX.25 ports configured on your Linux machine. For AX.25 ports the file that is read is the `/etc/ax25/axports` file. You must have an entry in this file for each AX.25 port you want on your system.

6.1. Creating the AX.25 network device

The network device is what is listed when you use the '*ifconfig*' command. This is the object that the Linux kernel sends and receives network data from. Nearly always the network device has a physical port associated with it, but there are occasions where this isn't necessary. The network device does relate directly to a device driver.

In the Linux AX.25 code there are a number of device drivers. The most common is probably the KISS driver, but others are the SCC driver(s), the Baycom driver and the Soundmodem driver.

Each of these device drivers will create a network device when it is started.

6.1.1. Creating a KISS device

Kernel Compile Options:

```
Amateur Radio support  --->
    [*] Amateur Radio support
    --- Packet Radio protocols
    <*>   Amateur Radio AX.25 Level 2 protocol
    ...
    AX.25 network device drivers  --->
    --- AX.25 network device drivers
    <*> Serial port KISS driver
    ...
```

Probably the most common configuration will be for a KISS TNC on a serial port. You will need to have the TNC preconfigured and connected to your serial port. You can use a communications program like *minicom* or *seyon* to configure the TNC into kiss mode.

To create a KISS device you use the *kissattach* program. In it simplest form you can use the *kissattach* program as follows:

```
# /usr/sbin/kissattach /dev/ttyS0 radio 44.135.96.242
# kissparms -p radio -t 100 -s 100 -r 25
```

The *kissattach* command will create a KISS network device. These devices are called '`ax[0-9]`'. The first time you use the *kissattach* command it creates '`ax0`', the second time it creates '`ax1`' etc. Each KISS device has an associated serial port.

The *kissparms* command allows you to set various KISS parameters on a KISS device.

Specifically the example presented would create a KISS network device using the serial device '`/dev/ttyS0`' and the entry from the `/etc/ax25/axports` with a port name of '`radio`'. It then configures it with a *txdelay* and *slottime* of 100 milliseconds and a *ppersist* value of 25.

Please refer to the *man* pages for more information.

6.1.1.1. Configuring for Dual Port TNC's

The *mkiss* utility included in the ax25-utils distribution allows you to make use of both modems on a dual port TNC. Configuration is fairly simple. It works by taking a single serial device connected to a

single multiport TNC and making it look like a number of devices each connected to a single port TNC. You do this *before* you do any of the AX.25 configuration. The devices that you then do the AX.25 configuration on are pseudo-TTY interfaces, (`/dev/ttyq*`), and not the actual serial device. Pseudo-TTY devices create a kind of pipe through which programs designed to talk to tty devices can talk to other programs designed to talk to tty devices. Each pipe has a master and a slave end. The master end is generally called '`/dev/ptyq*`' and the slave ends are called '`/dev/ttyq*`'. There is a one to one relationship between masters and slaves, so `/dev/ptyq0` is the master end of a pipe with `/dev/ttyq0` as its slave. You must open the master end of a pipe before opening the slave end. *mkiss* exploits this mechanism to split a single serial device into separate devices.

Example: if you have a dual port TNC and it is connected to your `/dev/ttyS0` serial device at 9600 bps, the command:

```
# /usr/sbin/mkiss -s 9600 /dev/ttyS0 /dev/ptyq0 /dev/ptyq1
# /usr/sbin/kissattach /dev/ttyq0 port1 44.135.96.242
# /usr/sbin/kissattach /dev/ttyq1 port2 44.135.96.242
```

would create two pseudo-tty devices that each look like a normal single port TNC. You would then treat `/dev/ttyq0` and `/dev/ttyq1` just as you would a conventional serial device with TNC connected. This means you'd then use the *kissattach* command as described above, on each of those, in the example for AX.25 ports called `port1` and `port2`. You shouldn't use *kissattach* on the actual serial device as the *mkiss* program uses it.

The *mkiss* command has a number of optional arguments that you may wish to use. They are summarized as follows:

-c

enables the addition of a one byte checksum to each KISS frame. This is not supported by most KISS implementations, it is supported by the G8BPG KISS ROM.

-s <speed>

sets the speed of the serial port.

-h

enables hardware handshaking on the serial port, it is off by default. Most KISS implementation do not support this, but some do.

-l

enables logging of information to the *syslog* log file.

6.1.2. Creating a 6PACK device

Kernel Compile Options:

```
Amateur Radio support  --->
    [*] Amateur Radio support
    --- Packet Radio protocols
    <*>   Amateur Radio AX.25 Level 2 protocol
    ...
    AX.25 network device drivers  --->
    --- AX.25 network device drivers
    ...
    <*> Serial port 6PACK driver
    ...
```

6PACK is a protocol that is supported by some TNCs as an alternative to KISS. It is used in a similar fashion to the KISS driver, using the `slattach` command instead of `kissattach`.

A mini HOWTO on the 6PACK driver is included in the kernel source code as the file `/usr/src/linux/Documentation/networking/6pack.txt`.

6.1.3. Creating a Baycom device

Kernel Compile Options:

```
Amateur Radio support  --->
    [*] Amateur Radio support
    --- Packet Radio protocols
    <*>   Amateur Radio AX.25 Level 2 protocol
    ...
    AX.25 network device drivers  --->
    --- AX.25 network device drivers
    ...
    <?> BAYCOM ser12 fullduplex driver for AX.25
    <?> BAYCOM ser12 halfduplex driver for AX.25
    <?> BAYCOM picpar and par96 driver for AX.25
    <?> BAYCOM epp driver for AX.25
    ...
```

Thomas Sailer (mailto:sailer@ife.ee.ethz.ch), despite the popularly held belief that it would not work very well, has developed Linux support for Baycom modems. His driver supports the `Ser12` serial port, `Par96` and the enhanced `PicPar` parallel port modems. Further information about the modems themselves may be obtained from the Baycom Web site (http://www.baycom.de/).

Your first step should be to determine the i/o and addresses of the serial or parallel port(s) you have Baycom modem(s) connected to. When you have these you must configure the Baycom driver with them.

The Baycom driver creates network devices called: `bc0`, `bc1`, `bc2` etc. when it is configured.

The *sethdlc* utility allows you to configure the driver with these parameters, or, if you have only one Baycom modem installed you may specify the parameters on the *insmod* command line when you load the Baycom module.

For example, a simple configuration. Disable the serial driver for COM1: then configure the Baycom driver for a Ser12 serial port modem on COM1: with the software DCD option enabled:

```
# setserial /dev/ttyS0 uart none
# insmod hdlcdrv
# insmod baycom mode="ser12*" iobase=0x3f8 irq=4
```

Par96 parallel port type modem on LPT1: using hardware DCD detection:

```
# insmod hdlcdrv
# insmod baycom mode="par96" iobase=0x378 irq=7 options=0
```

This is not really the preferred way to do it. The *sethdlc* utility works just as easily with one device as with many.

The *sethdlc man* page has the full details, but a couple of examples will illustrate the most important aspects of this configuration. The following examples assume you have already loaded the Baycom module using:

```
# insmod hdlcdrv
# insmod baycom
```

or that you compiled the kernel with the driver inbuilt.

Configure the `bc0` device driver as a Parallel port Baycom modem on LPT1: with software DCD:

```
# sethdlc -p -i bc0 mode par96 io 0x378 irq 7
```

Configure the `bc1` device driver as a Serial port Baycom modem on COM1:

```
# sethdlc -p -i bc1 mode "ser12*" io 0x3f8 irq 4
```

6.1.4. Configuring the AX.25 channel access parameters

The AX.25 channel access parameters are the equivalent of the KISS ppersist, txdelay and slottime type parameters. Again you use the *sethdlc* utility for this.

Again the *sethdlc* man page is the source of the most complete information but another example of two won't hurt:

Configure the `bc0` device with TxDelay of 200 mS, SlotTime of 100 mS, PPersist of 40 and half duplex:

```
# sethdlc -i bc0 -a txd 200 slot 100 ppersist 40 half
```

Note that the timing values are in milliseconds.

6.1.4.1. Configuring the Kernel AX.25 to use the Baycom device

The Baycom driver creates standard network devices that the AX.25 Kernel code can use. Configuration is much the same as that for a PI or PacketTwin card.

The first step is to configure the device with an AX.25 callsign. The *ifconfig* utility may be used to perform this.

```
# /sbin/ifconfig bc0 hw ax25 VK2KTJ-15 up
```

will assign the Baycom device `bc0` the AX.25 callsign `VK2KTJ-15`. Alternatively you can use the *axparms* command, you'll still need to use the *ifconfig* command to bring the device up though:

```
# ifconfig bc0 up
# axparms -setcall bc0 vk2ktj-15
```

The next step is to create an entry in the `/etc/ax25/axports` file as you would for any other device. The entry in the `axports` file is associated with the network device you've configured by the callsign you configure. The entry in the `axports` file that has the callsign that you configured the Baycom device with is the one that will be used to refer to it.

You may then treat the new AX.25 device as you would any other. You can configure it for TCP/IP, add it to ax25d and run NET/ROM or ROSE over it as you please.

6.1.5. Creating a kernel Soundmodem device

Kernel Compile Options:

```
Amateur Radio support  --->
    [*] Amateur Radio support
    --- Packet Radio protocols
    <*>   Amateur Radio AX.25 Level 2 protocol
    ...
    AX.25 network device drivers  --->
    --- AX.25 network device drivers
    ...
    <*> Soundcard modem driver
    [?]   soundmodem support for Soundblaster and compatible cards
    [?]   soundmodem support for WSS and Crystal cards
    [?]   soundmodem support for 1200 baud AFSK modulation
    [?]   soundmodem support for 2400 baud AFSK modulation (7.3728MHz crystal)
    [?]   soundmodem support for 2400 baud AFSK modulation (8MHz crystal)
    [?]   soundmodem support for 2666 baud AFSK modulation
    [?]   soundmodem support for 4800 baud HAPN-1 modulation
    [?]   soundmodem support for 4800 baud PSK modulation
    [?]   soundmodem support for 9600 baud FSK G3RUH modulation
    ...
```

Thomas Sailer has built a driver for the kernel that allows you to use your soundcard as a modem. Connect your radio directly to your soundcard to play packet! Thomas recommends at least a 486DX2/66 if you want to use this software as all of the digital signal processing is done by the main CPU.

The driver currently emulates 1200 bps AFSK, 4800 HAPN and 9600 FSK (G3RUH compatible) modem types. The only sound cards currently supported are SoundBlaster and Windows Sound System

Compatible models. If you have a sound card of another type, you can try the user-mode soundmodem described later in this document.

The sound cards require some circuitry to help them drive the Push-To-Talk circuitry, and information on this is available from Thomas's Soundmodem PTT circuit web page (http://www.baycom.org/~tom/pcf/ptt_circ/ptt.html). There are quite a few possible options, they are: detect the sound output from the soundcard, or use output from a parallel port, serial port or MIDI port. Circuit examples for each of these are on Thomas's site.

The Soundmodem driver creates network devices called: `sm0`, `sm1`, `sm2` etc when it is configured.

> **Note:** The Soundmodem driver competes for the same resources as the Linux sound driver, so if you wish to use the Soundmodem driver you must ensure that the Linux sound driver is not installed. You can, of course, compile them both as modules and insert and remove them as you wish.

6.1.5.1. Configuring the sound card

The Soundmodem driver does not initialize the sound card. The ax25-utils package includes a utility to do this called '*setcrystal*' that may be used for sound cards based on the Crystal chip set. If you have some other card then you will have to use some other software to initialize it. Its syntax is fairly straightforward:

```
setcrystal [-w wssio] [-s sbio] [-f synthio] [-i irq] [-d dma] [-c dma2]
```

So, for example, if you wished to configure a SoundBlaster card at i/o base address 0x388, irq 10 and DMA 1 you would use:

```
# setcrystal -s 0x388 -i 10 -d 1
```

To configure a Window Sound System card at i/o base address 0x534, irq 5, DMA 3 you would use:

```
# setcrystal -w 0x534 -i 5 -d 3
```

The `[-f synthio]` parameter is the set the synthesizer address, and the `[-c dma2]` parameter is to set the second DMA channel to allow full duplex operation.

6.1.5.2. Configuring the Soundmodem driver

When you have configured the soundcard you need to configure the driver telling it where the sound card is located and what sort of modem you wish it to emulate.

The *sethdlc* utility allows you to configure the driver with these parameters, or, if you have only one soundcard installed you may specify the parameters on the *insmod* command line when you load the Soundmodem module.

For example, a simple configuration, with one SoundBlaster soundcard configured as described above emulating a 1200 bps modem:

```
# insmod hdlcdrv
# insmod soundmodem mode="sbc:afsk1200" iobase=0x220 irq=5 dma=1
```

This is not really the preferred way to do it. The *sethdlc* utility works just as easily with one device as with many.

The *sethdlc* man page has the full details, but a couple of examples will illustrate the most important aspects of this configuration. The following examples assume you have already loaded the Soundmodem modules using:

```
# insmod hdlcdrv
# insmod soundmodem
```

or that you compiled the kernel with the driver inbuilt.

Configure the driver to support the Windows Sound System card we configured above to emulate a G3RUH 9600 compatible modem as device `sm0` using a parallel port at 0x378 to key the Push-To-Talk:

```
# sethdlc -p -i sm0 mode wss:fsk9600 io 0x534 irq 5 dma 3 pario 0x378
```

Configure the driver to support the SoundBlaster card we configured above to emulate a 4800 bps HAPN modem as device `sm1` using the serial port located at 0x2f8 to key the Push-To-Talk:

```
# sethdlc -p -i sm1 mode sbc:hapn4800 io 0x388 irq 10 dma 1 serio 0x2f8
```

Configure the driver to support the SoundBlaster card we configured above to emulate a 1200 bps AFSK modem as device `sm1` using the serial port located at 0x2f8 to key the Push-To-Talk:

```
# sethdlc -p -i sm1 mode sbc:afsk1200 io 0x388 irq 10 dma 1 serio 0x2f8
```

6.1.5.3. Configuring the AX.25 channel access parameters

The AX.25 channel access parameters are the equivalent of the KISS ppersist, txdelay and slottime type parameters. You use the *sethdlc* utility for this as well.

Again the *sethdlc* man page is the source of the most complete information but another example of two won't hurt:

Configure the `sm0` device with TxDelay of 100 mS, SlotTime of 50mS, PPersist of 128 and full duplex:

```
# sethdlc -i sm0 -a txd 100 slot 50 ppersist 128 full
```

Note that the timing values are in milliseconds.

6.1.5.4. Setting the audio levels and tuning the driver

It is very important that the audio levels be set correctly for any radio based modem to work. This is equally true of the Soundmodem. Thomas has developed some utility programs that make this task easier. They are called *smdiag* and *smmixer*.

smdiag

provides two types of display, either an oscilloscope type display or an eye pattern type display.

smmixer

allows you to actually adjust the transmit and receive audio levels.

To start the *smdiag* utility in 'eye' mode for the Soundmodem device `sm0` you would use:

```
# smdiag -i sm0 -e
```

To start the *smmixer* utility for the Soundmodem device `sm0` you would use:

```
# smmixer -i sm0
```

6.1.5.5. Configuring the Kernel AX.25 to use the Soundmodem

The Soundmodem driver creates standard network devices that the AX.25 Kernel code can use. Configuration is much the same as that for a PI or PacketTwin card.

The first step is to configure the device with an AX.25 callsign. The *ifconfig* utility may be used to perform this.

```
# /sbin/ifconfig sm0 hw ax25 VK2KTJ-15 up
```

will assign the Soundmodem device `sm0` the AX.25 callsign `VK2KTJ-15`. Alternatively you can use the *axparms* command, but you still need the *ifconfig* utility to bring the device up:

```
# ifconfig sm0 up
# axparms -setcall sm0 vk2ktj-15
```

The next step is to create an entry in the `/etc/ax25/axports` file as you would for any other device. The entry in the `axports` file is associated with the network device you've configured by the callsign you configure. The entry in the `axports` file that has the callsign that you configured the Soundmodem device with is the one that will be used to refer to it.

You may then treat the new AX.25 device as you would any other. You can configure it for TCP/IP, add it to ax25d and run NET/ROM or ROSE over it as you please.

6.1.6. Creating a user-mode Soundmodem device

Kernel Compile Options: not applicable

Thomas Sailer has written a sound modem driver that runs in user-mode using the kernel sound drivers, so it should work with any sound card supported under Linux.

The driver is implemented as the user-mode program `soundmodem`. The graphical `soundmodemconfig` program allows configuring and testing the soundmodem driver. As well as kernel sound support you need the kernel AX.25 mkiss driver.

The software and documentation can be downloaded from http://www.baycom.org/~tom/ham/soundmodem (http://www.baycom.org/~tom/ham/soundmodem/).

6.1.7. Creating a YAM device

Kernel Compile Options:

```
Amateur Radio support  --->
    [*] Amateur Radio support
    --- Packet Radio protocols
    <*>   Amateur Radio AX.25 Level 2 protocol
    ...
    AX.25 network device drivers  --->
    --- AX.25 network device drivers
    ...
    <?> YAM driver for AX.25
    ...
```

YAM is Yet Another Modem, a 9600 baud modem designed by Nico Palermo. Information on the Linux driver can be found at http://www.teaser.fr/~frible/yam.html while general information on the modem can be found at http://www.microlet.com/yam/

6.1.8. Creating a PI card device

Kernel Compile Options:

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] Ottawa PI and PI/2 support for AX.25
```

The PI card device driver creates devices named '`pi[0-9][ab]`'. The first PI card detected will be allocated '`pi0`', the second '`pi1`' etc. The '`a`' and '`b`' refer to the first and second physical interface on the PI card. If you have built your kernel to include the PI card driver, and the card has been properly detected then you can use the following command to configure the network device:

```
# /sbin/ifconfig pi0a hw ax25 VK2KTJ-15 up
```

This command would configure the first port on the first PI card detected with the callsign `VK2KTJ-15` and make it active. To use the device all you now need to do is to configure an entry into your `/etc/ax25/axports` file with a matching callsign/ssid and you will be ready to continue on.

The PI card driver was written by David Perry (mailto:dp@hydra.carleton.edu).

6.1.9. Creating a PacketTwin device

Kernel Compile Options:

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] Gracilis PackeTwin support for AX.25
```

The PacketTwin card device driver creates devices named '`pt[0-9][ab]`'. The first PacketTwin card detected will be allocated '`pt0`', the second '`pt1`' etc. The '`a`' and '`b`' refer to the first and second

physical interface on the PacketTwin card. If you have built your kernel to include the PacketTwin card driver, and the card has been properly detected then you can use the following command to configure the network device:

```
# /sbin/ifconfig pt0a hw ax25 VK2KTJ-15 up
```

This command would configure the first port on the first PacketTwin card detected with the callsign `VK2KTJ-15` and make it active. To use the device all you now need to do is to configure an entry into your `/etc/ax25/axports` file with a matching callsign/ssid and you will be ready to continue on.

The PacketTwin card driver was written by Craig Small (mailto:csmall@triode.apana.org.au), VK2XLZ.

6.1.10. Creating a generic SCC device

Kernel Compile Options:

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] Z8530 SCC KISS emulation driver for AX.25
```

Joerg Reuter (mailto:jreuter@poboxes.com), DL1BKE, has developed generic support for Z8530 SCC based cards. His driver is configurable to support a range of different types of cards and present an interface that looks like a KISS TNC so you can treat it as though it were a KISS TNC.

6.1.10.1. Obtaining and building the configuration tool package

While the kernel driver is included in the standard kernel distribution, Joerg distributes more recent versions of his driver with the suite of configuration tools that you will need to obtain as well.

You can obtain the configuration tools package from: Joerg's web page (http://www.qsl.net/dl1bke), ftp://db0bm.automation.fh-aachen.de/incoming/dl1bke, ftp://insl1.etec.uni-karlsruhe.de/pub/hamradio/linux/z8530, ftp://ftp.ucsd.edu/hamradio/packet/tcpip/linux, or ftp://ftp.ucsd.edu/hamradio/packet/tcpip/incoming.

You will find multiple versions, choose the one that best suits the kernel you intend to use: `z8530drv-2.4a.dl1bke.tar.gz` for 2.0.* kernels and `z8530drv-utils-3.0.tar.gz` for 2.1.6 or later kernels.

The following commands were what I used to compile and install the package for kernel version 2.0.30:

```
# cd /usr/src
# gzip -dc z8530drv-2.4a.dl1bke.tar.gz | tar xvpofz -
# cd z8530drv
# make clean
# make dep
# make module         # If you want to build the driver as a module
# make for_kernel     # If you want the driver to built into your kernel
# make install
```

After the above is complete you should have three new programs installed in your `/sbin` directory: *gencfg*, *sccinit* and *sccstat*. It is these programs that you will use to configure the driver for your card.

You will also have a group of new special device files created in your `/dev` called `scc0-scc7`. These will be used later and will be the 'KISS' devices you will end up using.

If you chose to 'make for_kernel' then you will need to recompile your kernel. To ensure that you include support for the z8530 driver you must be sure to answer '`Y`' to: '`Z8530 SCC kiss emulation driver for AX.25`' when asked during a kernel '`make config`'.

If you chose to 'make module' then the new `scc.o` will have been installed in the appropriate `/lib/modules` directory and you do not need to recompile your kernel. Remember to use the *insmod* command to load the module before your try and configure it.

6.1.10.2. Configuring the driver for your card

The z8530 SCC driver has been designed to be as flexible as possible so as to support as many different types of cards as possible. With this flexibility has come some cost in configuration.

There is more comprehensive documentation in the package and you should read this if you have any problems. You should particularly look at `doc/scc_eng.doc` or `doc/scc_ger.doc` for more detailed information. I've paraphrased the important details, but as a result there is a lot of lower level detail that I have not included.

The main configuration file is read by the *sccinit* program and is called `/etc/z8530drv.conf`. This file is broken into two main stages: Configuration of the hardware parameters and channel configuration. After you have configured this file you need only add:

```
# sccinit
```

into the `rc` file that configures your network and the driver will be initialized according to the contents of the configuration file. You must do this before you attempt to use the driver.

6.1.10.2.1. Configuration of the hardware parameters

The first section is broken into stanzas, each stanza representing an 8530 chip. Each stanza is a list of keywords with arguments. You may specify up to four SCC chips in this file by default. The `#define MAXSCC 4` in `scc.c` can be increased if you require support for more.

The allowable keywords and arguments are:

chip

: the `chip` keyword is used to separate stanzas. It will take anything as an argument. The arguments are not used.

data_a

: this keyword is used to specify the address of the data port for the z8530 channel ‘A’. The argument is a hexadecimal number e.g. 0x300

ctrl_a

: this keyword is used to specify the address of the control port for the z8530 channel ‘A’. The arguments is a hexadecimal number e.g. 0x304

data_b

: this keyword is used to specify the address of the data port for the z8530 channel ‘B’. The argument is a hexadecimal number e.g. 0x301

ctrl_b

: this keyword is used to specify the address of the control port for the z8530 channel ‘B’. The arguments is a hexadecimal number e.g. 0x305

irq

: this keyword is used to specify the IRQ used by the 8530 SCC described in this stanza. The argument is an integer e.g. 5

pclock

this keyword is used to specify the frequency of the clock at the PCLK pin of the 8530. The argument is an integer frequency in Hz which defaults to 4915200 if the keyword is not supplied.

board

the type of board supporting this 8530 SCC. The argument is a character string. The allowed values are:

PA0HZP

the PA0HZP SCC Card

EAGLE

the Eagle card

PC100

the DRSI PC100 SCC card

PRIMUS

the PRIMUS-PC (DG9BL) card

BAYCOM

BayCom (U)SCC card

escc

this keyword is optional and is used to enable support for the Extended SCC chips (ESCC) such as the 8580, 85180, or the 85280. The argument is a character string with allowed values of ‘yes’ or ‘no’. The default is ‘no’.

vector

this keyword is optional and specifies the address of the vector latch (also known as "intack port") for PA0HZP cards. There can be only one vector latch for all chips. The default is 0.

special

this keyword is optional and specifies the address of the special function register on several cards. The default is 0.

option

this keyword is optional and defaults to 0.

Some example configurations for the more popular cards are as follows:

BayCom USCC

```
chip    1
data_a  0x300
ctrl_a  0x304
data_b  0x301
ctrl_b  0x305
irq     5
board   BAYCOM
#
# SCC chip 2
#
chip    2
data_a  0x302
ctrl_a  0x306
data_b  0x303
ctrl_b  0x307
board   BAYCOM
```

PA0HZP SCC card

```
chip 1
data_a 0x153
data_b 0x151
ctrl_a 0x152
ctrl_b 0x150
irq 9
pclock 4915200
board PA0HZP
vector 0x168
escc no
#
#
#
chip 2
data_a 0x157
data_b 0x155
ctrl_a 0x156
ctrl_b 0x154
irq 9
pclock 4915200
board PA0HZP
vector 0x168
escc no
```

DRSI SCC card

```
chip 1
data_a 0x303
data_b 0x301
```

```
ctrl_a 0x302
ctrl_b 0x300
irq 7
pclock 4915200
board DRSI
escc no
```

If you already have a working configuration for your card under NOS, then you can use the *gencfg* command to convert the PE1CHL NOS driver commands into a form suitable for use in the z8530 driver configuration file.

To use *gencfg* you simply invoke it with the same parameters as you used for the PE1CHL driver in NET/NOS. For example:

```
# gencfg 2 0x150 4 2 0 1 0x168 9 4915200
```

will generate a skeleton configuration for the OptoSCC card.

6.1.10.3. Channel Configuration

The Channel Configuration section is where you specify all of the other parameters associated with the port you are configuring. Again this section is broken into stanzas. One stanza represents one logical port, and therefore there would be two of these for each one of the hardware parameters stanzas as each 8530 SCC supports two ports.

These keywords and arguments are also written to the `/etc/z8530drv.conf` file and must appear *after* the hardware parameters section.

Sequence is very important in this section, but if you stick with the suggested sequence it should work okay. The keywords and arguments are:

device

this keyword must be the first line of a port definition and specifies the name of the special device file that the rest of the configuration applies to. e.g. `/dev/scc0`

speed

this keyword specifies the speed in bits per second of the interface. The argument is an integer: e.g. `1200`

clock

this keyword specifies where the clock for the data will be sourced. Allowable values are:

dpll

normal halfduplex operation

external

MODEM supplies its own Rx/Tx clock

divider

use fullduplex divider if installed.

mode

this keyword specifies the data coding to be used. Allowable arguments are: `nrzi` or `nrz`

rxbuffers

this keyword specifies the number of receive buffers to allocate memory for. The argument is an integer, e.g. 8.

txbuffers

this keyword specifies the number of transmit buffers to allocate memory for. The argument is an integer, e.g. 8.

bufsize

this keyword specifies the size of the receive and transmit buffers. The arguments is in bytes and represents the total length of the frame, so it must also take into account the AX.25 headers and not just the length of the data field. This keyword is optional and default to `384`

txdelay

the KISS transmit delay value, the argument is an integer in mS.

persist

the KISS persist value, the argument is an integer.

slot

the KISS slot time value, the argument is an integer in mS.

tail

the KISS transmit tail value, the argument is an integer in mS.

fulldup

the KISS full duplex flag, the argument is an integer. `1`==Full Duplex, `0`==Half Duplex.

wait

the KISS wait value, the argument is an integer in mS.

min

the KISS min value, the argument is an integer in S.

maxkey

the KISS maximum keyup time, the argument is an integer in S.

idle

the KISS idle timer value, the argument is an integer in S.

maxdef

the KISS maxdef value, the argument is an integer.

group

the KISS group value, the argument is an integer.

txoff

the KISS txoff value, the argument is an integer in mS.

softdcd

the KISS softdcd value, the argument is an integer.

slip

the KISS slip flag, the argument is an integer.

6.1.10.4. Using the driver

To use the driver you simply treat the `/dev/scc*` devices just as you would a serial tty device with a KISS TNC connected to it. For example, to configure Linux Kernel networking to use your SCC card you could use something like:

```
# kissattach -s 4800 /dev/scc0 VK2KTJ
```

You can also use NOS to attach to it in precisely the same way. From JNOS for example you would use something like:

```
attach asy scc0 0 ax25 scc0 256 256 4800
```

6.1.10.5. The sccstat and sccparam tools

To assist in the diagnosis of problems you can use the *sccstat* program to display the current configuration of an SCC device. To use it try:

```
# sccstat /dev/scc0
```

you will displayed a very large amount of information relating to the configuration and health of the `/dev/scc0` SCC port.

The *sccparam* command allows you to change or modify a configuration after you have booted. Its syntax is very similar to the NOS `param` command, so to set the `txtail` setting of a device to 100mS you would use:

```
# sccparam /dev/scc0 txtail 0x8
```

6.1.11. Creating a BPQ ethernet device

Kernel Compile Options:

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] BPQ Ethernet driver for AX.25
```

Linux supports BPQ Ethernet compatibility. This enables you to run the AX.25 protocol over your Ethernet LAN and to interwork your linux machine with other BPQ machines on the LAN.

The BPQ network devices are named '`bpq[0-9]`'. The '`bpq0`' device is associated with the '`eth0`' device, the '`bpq1`' device with the '`eth1`' device etc.

Configuration is quite straightforward. You firstly must have configured a standard Ethernet device. This means you will have compiled your kernel to support your Ethernet card and tested that this works. Refer to the Ethernet-HOWTO (Ethernet-HOWTO.html) for more information on how to do this.

To configure the BPQ support you need to configure the Ethernet device with an AX.25 callsign. The following command will do this for you:

```
# /sbin/ifconfig bpq0 hw ax25 vk2ktj-14 up
```

Again, remember that the callsign you specify should match the entry in the `/etc/ax25/axports` file that you wish to use for this port.

6.1.12. Configuring the BPQ Node to talk to the Linux AX.25 support

BPQ Ethernet normally uses a multicast address. The Linux implementation does not, and instead it uses the normal Ethernet broadcast address. The NET.CFG file for the BPQ ODI driver should therefore be modified to look similar to this:

```
LINK SUPPORT

        MAX STACKS 1
        MAX BOARDS 1

LINK DRIVER E2000                   ; or other MLID to suit your card

        INT 10                      ;
        PORT 300                    ; to suit your card

        FRAME ETHERNET_II

        PROTOCOL BPQ 8FF ETHERNET_II ; required for BPQ - can change PID

BPQPARAMS                           ; optional - only needed if you want
                                    ; to override the default target addr

        ETH_ADDR  FF:FF:FF:FF:FF:FF  ; Target address
```

6.2. Creating the `/etc/ax25/axports` file

The `/etc/ax25/axports` is a simple text file that you create with a text editor. The format of the `/etc/ax25/axports` file is as follows:

```
portname  callsign  baudrate  paclen  window  description
```

where:

portname
: is a text name that you will refer to the port by.

callsign
: is the AX.25 callsign you want to assign to the port.

baudrate
: is the speed at which you wish the port to communicate with your TNC.

paclen
: is the maximum packet length you want to configure the port to use for AX.25 connected mode connections.

window
: is the AX.25 window (K) parameter. This is the same as the `MAXFRAME` setting of many TNC's.

description
: is a textual description of the port.

In my case, mine looks like:

```
radio    VK2KTJ-15       4800        256     2       4800bps 144.800 MHz
ether    VK2KTJ-14       10000000    256     2       BPQ/ethernet device
```

Remember, you must assign unique callsign/ssid to each AX.25 port you create. Create one entry for each AX.25 device you want to use, this includes KISS, Baycom, SCC, PI, PT and Soundmodem ports.

Each entry here will describe exactly one AX.25 network device. The entries in this file are associated with the network devices by the callsign/ssid. This is at least one good reason for requiring unique callsign/ssid.

6.3. Configuring AX.25 routing

You may wish to configure default digipeaters paths for specific hosts. This is useful for both normal AX.25 connections and also IP based connections. The *axparms* command enables you to do this. Again, the *man* page offers a complete description, but a simple example might be:

```
# /usr/sbin/axparms -route add radio VK2XLZ VK2SUT
```

This command would set a digipeater entry for `VK2XLZ` via `VK2SUT` on the AX.25 port named `radio`.

7. Configuring an AX.25 interface for TCP/IP

It is very simple to configure an AX.25 port to carry TCP/IP. If you have KISS interfaces then there are two methods for configuring an IP address. The *kissattach* command has an option that allows you to specify an IP address. The more conventional method using the *ifconfig* command will work on all interface types.

So, modifying the previous KISS example:

```
# /usr/sbin/kissattach -i 44.136.8.5 -m 512 /dev/ttyS0 radio
# /sbin/route add -net 44.136.8.0 netmask 255.255.255.0 ax0
# /sbin/route add default ax0
```

to create the AX.25 interface with an IP address of `44.136.8.5` and an `MTU` of `512` bytes. You should still use the *ifconfig* to configure the other parameters if necessary.

If you have any other interface type then you use the *ifconfig* program to configure the ip address and netmask details for the port and add a route via the port, just as you would for any other TCP/IP interface. The following example is for a PI card device, but would work equally well for any other AX.25 network device:

```
# /sbin/ifconfig pi0a 44.136.8.5 netmask 255.255.255.0 up
# /sbin/ifconfig pi0a broadcast 44.136.8.255 mtu 512
# /sbin/route add -net 44.136.8.0 netmask 255.255.255.0 pi0a
# /sbin/route add default pi0a
```

The commands listed above are typical of the sort of configuration many of you would be familiar with if you have used NOS or any of its derivatives or any other TCP/IP software. Note that the default route might not be required in your configuration if you have some other network device configured.

To test it out, try a ping or a telnet to a local host.

```
# ping -i 5 44.136.8.58
```

Note the use of the '`-i 5`' arguments to *ping* to tell it to send pings every 5 seconds instead of its default of 1 second.

8. Configuring a NET/ROM port

The NET/ROM protocol relies on, and uses the AX.25 ports you have created. The NET/ROM protocol rides on top of the AX.25 protocol. To configure NET/ROM on an AX.25 interface you must configure two files. One file describes the NET/ROM interfaces, and the other file describes which of the AX.25 ports will carry NET/ROM. You can configure multiple NET/ROM ports, each with its own callsign and alias, the same procedure applies for each.

8.1. Configuring `/etc/ax25/nrports`

The first is the `/etc/ax25/nrports` file. This file describes the NET/ROM ports in much the same way as the `/etc/ax25/axports` file describes the AX.25 ports. Each NET/ROM device you wish to create must have an entry in the `/etc/ax25/nrports` file. Normally a Linux machine would have only one NET/ROM device configured that would use a number of the AX.25 ports defined. In some situations you might wish a special service such as a BBS to have a separate NET/ROM alias and so you would create more than one.

This file is formatted as follows:

```
name callsign  alias  paclen   description
```

Where:

name

is the text name that you wish to refer to the port by.

callsign

is the callsign that the NET/ROM traffic from this port will use. Note, this is *not* that address that users should connect to to get access to a *node* style interface. (The node program is covered later). This callsign/ssid should be unique and should not appear elsewhere in either of the `/etc/ax25/axports` or the `/etc/ax25/nrports` files.

alias

is the NET/ROM alias this port will have assigned to it.

paclen

is the maximum size of NET/ROM frames transmitted by this port.

description

is a free text description of the port.

An example would look something like the following:

```
netrom  VK2KTJ-9        LINUX   236     Linux Switch Port
```

This example creates a NET/ROM port known to the rest of the NET/ROM network as '`LINUX:VK2KTJ-9`'.

This file is used by programs such as the *call* program.

8.2. Configuring `/etc/ax25/nrbroadcast`

The second file is the `/etc/ax25/nrbroadcast` file. This file may contain a number of entries. There would normally be one entry for each AX.25 port that you wish to allow NET/ROM traffic on.

This file is formatted as follows:

```
axport min_obs def_qual worst_qual verbose
```

Where:

axport

is the port name obtained from the `/etc/ax25/axports` file. If you do not have an entry in `/etc/ax25/nrbroadcasts` for a port then this means that no NET/ROM routing will occur and any received NET/ROM broadcasts will be ignored for that port.

min_obs

is the minimum obselesence value for the port.

def_qual

is the default quality for the port.

worst_qual

is the worst quality value for the port, any routes under this quality will be ignored.

verbose

is a flag determining whether full NET/ROM routing broadcasts will occur from this port or only a routing broadcast advertising the node itself.

An example would look something like the following:

```
radio    1       200      100         1
```

8.3. Creating the NET/ROM Network device

When you have the two configuration files completed you must create the NET/ROM device in much the same way as you did for the AX.25 devices. This time you use the *nrattach* command. The *nrattach* works in just the same way as the *axattach* command except that it creates NET/ROM network devices called '`nr[0-9]`'. Again, the first time you use the *nrattach* command it creates the '`nr0`' device, the second time it creates the '`nr1`' network devices etc. To create the network device for the NET/ROM port we've defined we would use:

```
# nrattach netrom
```

This command would start the NET/ROM device (`nr0`) named `netrom` configured with the details specified in the `/etc/ax25/nrports` file.

8.4. Starting the NET/ROM daemon

The Linux kernel does all of the NET/ROM protocol and switching, but does not manage some functions. The NET/ROM daemon manages the NET/ROM routing tables and generates the NET/ROM routing broadcasts. You start NET/ROM daemon with the command:

```
# /usr/sbin/netromd -i
```

You should soon see the `/proc/net/nr_neigh` file filling up with information about your NET/ROM neighbours.

Remember to put the `/usr/sbin/netromd` command in your *rc* files so that it is started automatically each time you reboot.

8.5. Configuring NET/ROM routing.

You may wish to configure static NET/ROM routes for specific hosts. The *nrparms* command enables you to do this. Again, the *man* page offers a complete description, but a simple example might be:

```
# /usr/sbin/nrparms -nodes VK2XLZ-10 + #MINTO 120 5 radio VK2SUT-9
```

This command would set a NET/ROM route to `#MINTO:VK2XLZ-10` via a neighbour `VK2SUT-9` on my AX.25 port called '`radio`'.

You can manually create entries for new neighbours using the *nrparms* command as well. For example:

```
# /usr/sbin/nrparms -routes radio VK2SUT-9 + 120
```

This command would create `VK2SUT-9` as a NET/ROM neighbour with a quality of `120` and this will be locked and will not be deleted automatically.

9. Configuring a NET/ROM interface for TCP/IP

Configuring a NET/ROM interface for TCP/IP is almost identical to configuring an AX.25 interface for TCP/IP.

Again you can either specify the ip address and mtu on the *nrattach* command line, or use the *ifconfig* and *route* commands, but you need to manually add *arp* entries for hosts you wish to route to because there is no mechanism available for your machine to learn what NET/ROM address it should use to reach a particular IP host.

So, to create an `nr0` device with an IP address of `44.136.8.5`, an mtu of `512` and configured with the details from the `/etc/ax25/nrports` file for a NET/ROM port named `netrom` you would use:

```
# /usr/sbin/nrattach -i 44.136.8.5 -m 512 netrom
# route add 44.136.8.5 nr0
```

or you could use something like the following commands manually:

```
# /usr/sbin/nrattach netrom
# ifconfig nr0 44.136.8.5 netmask 255.255.255.0 hw netrom VK2KTJ-9
# route add 44.136.8.5 nr0
```

Then for each IP host you wish to reach via NET/ROM you need to set route and arp entries. To reach a destination host with an IP address of `44.136.80.4` at NET/ROM address `BBS:VK3BBS` via a NET/ROM neighbour with callsign `VK2SUT-0` you would use commands as follows:

```
# route add 44.136.80.4 nr0
# arp -t netrom -s 44.136.80.4 vk2sut-0
# nrparms -nodes vk3bbs + BBS 120 6 sl0 vk2sut-0
```

The '`120`' and '`6`' arguments to the *nrparms* command are the NET/ROM *quality* and *obsolescence count* values for the route.

10. Configuring a ROSE port

The ROSE packet layer protocol is similar to layer three of the X.25 specification. The kernel based ROSE support is a *modified* version of the FPAC Rose implementation (http://fpac.lmi.ecp.fr/f1oat/f1oat.html).

The ROSE packet layer protocol protocol relies on, and uses the AX.25 ports you have created. The ROSE protocol rides on top of the AX.25 protocol. To configure ROSE you must create a configuration file that describes the ROSE ports you want. You can create multiple ROSE ports if you wish, the same procedure applies for each.

10.1. Configuring `/etc/ax25/rsports`

The file where you configure your ROSE interfaces is the `/etc/ax25/rsports` file. This file describes the ROSE port in much the same way as the `/etc/ax25/axports` file describes the AX.25 ports.

This file is formatted as follows:

```
name  address  description
```

Where:

name
: is the text name that you wish to refer to the port by.

address
: is the 10 digit ROSE address you wish to assign to this port.

description
: is a free text description of the port.

An example would look something like the following:

```
rose  5050294760  Rose Port
```

Note that ROSE will use the default callsign/ssid configured on each AX.25 port unless you specify otherwise.

To configure a separate callsign/ssid for ROSE to use on each port you use the *rsparms* command as follows:

```
# /usr/sbin/rsprams -call VK2KTJ-10
```

This example would make Linux listen for and use the callsign/ssid `VK2KTJ-10` on all of the configured AX.25 ports for ROSE calls.

10.2. Creating the ROSE Network device

When you have created the `/etc/ax25/rsports` file you may create the ROSE device in much the same way as you did for the AX.25 devices. This time you use the *rsattach* command. The *rsattach* command creates network devices named '`rose[0-5]`'. The first time you use the *rsattach* command it create the '`rose0`' device, the second time it creates the '`rose1`' device etc. For example:

```
# rsattach rose
```

This command would start the ROSE device (`rose0`) configured with the details specified in the `/etc/ax25/rsports` file for the entry named '`rose`'.

10.3. Configuring ROSE Routing

The ROSE protocol currently supports only static routing. The *rsparms* utility allows you to configure your ROSE routing table under Linux.

For example:

```
# rsparms -nodes add 5050295502 radio vk2xlz
```

would add a route to ROSE node `5050295502` via an AX.25 port named '`radio`' in your `/etc/ax25/axports` file to a neighbour with the callsign `VK2XLZ`.

You may specify a route with a mask to capture a number of ROSE destinations into a single routing entry. The syntax looks like:

```
# rsparms -nodes add 5050295502/4 radio vk2xlz
```

which would be identical to the previous example except that it would match any destination address that matched the first four digits supplied, in this case any address commencing with the digits `5050`. An alternate form for this command is:

```
# rsparms -nodes add 5050/4 radio vk2xlz
```

which is probably the less ambiguous form.

11. Making AX.25/NET/ROM/ROSE calls

Now that you have all of your AX.25, NET/ROM and ROSE interfaces configured and active, you should be able to make test calls.

The AX.25 Utilities package includes a program called '*call*' which is a split screen terminal program for AX.25, NET/ROM and ROSE.

A simple AX.25 call would look like:

```
/usr/bin/call radio VK2DAY via VK2SUT
```

A simple NET/ROM call to a node with an alias of `SUNBBS` would look like:

```
/usr/bin/call netrom SUNBBS
```

A simple ROSE call to `HEARD` at node `5050882960` would look like:

```
/usr/bin/call rose HEARD 5050882960
```

Note: you must tell *call* which port you wish to make the call on, as the same destination node might be reachable on any of the ports you have configured.

The *call* program is a line mode terminal program for making AX.25 calls. It recognizes lines that start with '~' as command lines. The '~.' command will close the connection.

Please refer to the man page in `/usr/man` for more information.

12. Configuring Linux to accept Packet connections

Linux is a powerful operating system and offers a great deal of flexibility in how it is configured. With this flexibility comes a cost in configuring it to do what you want. When configuring your Linux machine to accept incoming AX.25, NET/ROM or ROSE connections there are a number of questions you need to ask yourself. The most important of which is: "What do I want users to see when they connect?". People are developing neat little applications that may be used to provide services to callers, a simple example is the *pms* program included in the AX.25 utilities, a more complex example is the *node* program also included in the AX.25 utilities. Alternatively you might want to give users a login prompt so that they can make use of a shell account, or you might even have written your own program, such as a customized database or a game, that you want people to connect to. Whatever you choose, you must tell the AX.25 software about this so that it knows what software to run when it accepts an incoming AX.25 connection.

The *ax25d* program is similar to the *inetd* program commonly used to accept incoming TCP/IP connections on UNIX machines. It sits and listens for incoming connections, when it detects one it goes away and checks a configuration file to determine what program to run and connect to that connection. Since this the standard tool for accepting incoming AX.25, NET/ROM and ROSE connections I'll describe how to configure it.

12.1. Creating the `/etc/ax25/ax25d.conf` file

This file is the configuration file for the *ax25d* AX.25 daemon which handles incoming AX.25, NET/ROM and ROSE connections.

The file is a little cryptic looking at first, but you'll soon discover it is very simple in practice, with a small trap for you to be wary of.

The general format of the `ax25d.conf` file is as follows:

```
# This is a comment and is ignored by the ax25d program.
[port_name] || <port_name> || {port_name}
<peer1>    window T1 T2 T3 idle N2 <mode> <uid> <cmd> <cmd-name> <arguments>
<peer2>    window T1 T2 T3 idle N2 <mode> <uid> <cmd> <cmd-name> <arguments>
parameters window T1 T2 T3 idle N2 <mode>
<peer3>    window T1 T2 T3 idle N2 <mode> <uid> <cmd> <cmd-name> <arguments>
   ...
default    window T1 T2 T3 idle N2 <mode> <uid> <cmd> <cmd-name> <arguments>
```

Where:

#

at the start of a line marks a comment and is completely ignored by the *ax25d* program.

<port_name>

is the name of the AX.25, NET/ROM or ROSE port as specified in the `/etc/ax25/axports`, `/etc/ax25/nrports` and `/etc/ax25/rsports` files. The name of the port is surrounded by the '`[]`' brackets if it is an AX.25 port, the '<>' brackets if it is a NET/ROM port, or the '{}' brackets if it is a ROSE port. There is an alternate form for this field, and that is use prefix the port name with '`callsign/ssid via`' to indicate that you wish accept calls to the callsign/ssid via this interface. The example should more clearly illustrate this.

<peer>

is the callsign of the peer node that this particular configuration applies to. If you don't specify an SSID here then any SSID will match.

window

is the AX.25 Window parameter (K) or MAXFRAME parameter for this configuration.

T1

is the Frame retransmission (T1) timer in half second units.

T2

is the amount of time the AX.25 software will wait for another incoming frame before preparing a response in 1 second units.

T3

is the amount of time of inactivity before the AX.25 software will disconnect the session in 1 second units.

idle

is the idle timer value in seconds.

N2

is the number of consecutive retransmissions that will occur before the connection is closed.

<mode>

provides a mechanism for determining certain types of general permissions. The modes are enabled or disabled by supplying a combination of characters, each representing a permission. The characters may be in either upper or lower case and must be in a single block with no spaces.

u/U

UTMP - currently unsupported.

v/V

Validate call - currently unsupported.

q/Q

Quiet - Don't log connection

n/N

check NET/ROM Neighbour - currently unsupported.

d/D

Disallow Digipeaters - Connections must be direct, not digipeated.

l/L

Lockout - Don't allow connection.

*/0

marker - place marker, no mode set.

<uid>

is the userid that the program to be run to support the connection should be run as.

<cmd>

is the full pathname of the command to be run, with no arguments specified.

<cmd-name>

is the text that should appear in a *ps* as the command name running (normally the same as <cmd> except without the directory path information.

<arguments>

are the command line argument to be passed to the <:cmd> when it is run. You pass useful information into these arguments by use of the following tokens:

%d

Name of the port the connection was received on.

%U

AX.25 callsign of the connected party without the SSID, in uppercase.

%u

AX.25 callsign of the connected party without the SSID, in lowercase.

%S

AX.25 callsign of the connected party with the SSID, in uppercase.

%s

AX.25 callsign of the connected party with the SSID, in lowercase.

%P

AX.25 callsign of the remote node that the connection came in from without the SSID, in uppercase.

%p

AX.25 callsign of the remote node that the connection came in from without the SSID, in lowercase.

%R

AX.25 callsign of the remote node that the connection came in from with the SSID, in uppercase.

%r

AX.25 callsign of the remote node that the connection came in from with the SSID, in lowercase.

You need one section in the above format for each AX.25, NET/ROM or ROSE interface you want to accept incoming AX.25, NET/ROM or ROSE connections on.

There are two special lines in the paragraph, one starts with the string '`parameters`' and the other starts with the string '`default`' (yes there is a difference). These lines serve special functions.

The ‘`default`’ lines purpose should be obvious, this line acts as a catch-all, so that any incoming connection on the <interface_call> interface that doesn’t have a specific rule will match the ‘`default`’ rule. If you don’t have a ‘`default`’ rule, then any connections not matching any specific rule will be disconnected immediately without notice.

The ‘`parameters`’ line is a little more subtle, and here is the trap I mentioned earlier. In any of the fields for any definition for a peer you can use the ‘*’ character to say ‘use the default value’. The ‘`parameters`’ line is what sets those default values. The kernel software itself has some defaults which will be used if you don’t specify any using the ‘`parameters`’ entry. The trap is that the these defaults apply *only* to those rules *below* the ‘`parameters`’ line, not to those above. You may have more than one ‘`parameters`’ rule per interface definition, and in this way you may create groups of default configurations. It is important to note that the ‘`parameters`’ rule does not allow you to set the ‘`uid`’ or ‘`command`’ fields.

12.2. A simple example `ax25d.conf` file

Okay, an illustrative example:

```
# ax25d.conf for VK2KTJ - 02/03/97
# This configuration uses the AX.25 port defined earlier.

# <peer> Win T1  T2  T3  idl N2 <mode> <uid> <exec> <argv[0]>[<args....>]

[VK2KTJ-0 via radio]
parameters 1    10  *  *  *   *   *
VK2XLZ     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
VK2DAY     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
NOCALL     *     *  *  *  *   *   L
default    1    10  5 100 180 5   *    root  /usr/sbin/pms pms -a -o vk2ktj

[VK2KTJ-1 via radio]
default    *     *    *   *   *   0    root /usr/sbin/node node

<netrom>
parameters 1    10  *  *  *   *   *
NOCALL     *     *  *  *  *   *   L
default    *     *  *  *  *   *   0        root /usr/sbin/node node

{VK2KTJ-0 via rose}
parameters 1    10  *  *  *   *   *
VK2XLZ     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
VK2DAY     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
NOCALL     *     *  *  *  *   *   L
default    1    10  5 100 180 5   *    root  /usr/sbin/pms pms -a -o vk2ktj

{VK2KTJ-1 via rose}
default    *     *    *   *   *   0    root /usr/sbin/node node radio
```

This example says that anybody attempting to connect to the callsign '`VK2KTJ-0`' heard on the AX.25 port called '`radio`' will have the following rules applied:

Anyone whose callsign is set to 'NOCALL' should be locked out, note the use of mode 'L'.

The `parameters` line changes two parameters from the kernel defaults (Window and T1) and will run the */usr/sbin/axspawn* program for them. Any copies of */usr/sbin/axspawn* run this way will appear as *axspawn* in a *ps* listing for convenience. The next two lines provide definitions for two stations who will receive those permissions.

The last line in the paragraph is the 'catch all' definition that everybody else will get (including VK2XLZ and VK2DAY using any other SSID other than -1). This definition sets all of the parameters implicitly and will cause the *pms* program to be run with a command line argument indicating that it is being run for an AX.25 connection, and that the owner callsign is `VK2KTJ`. (See the 'Configuring the PMS' section below for more details).

The next configuration accepts calls to `VK2KTJ-1` via the `radio` port. It runs the *node* program for everybody that connects to it.

The next configuration is a NET/ROM configuration, note the use of the greater-then and less-than braces instead of the square brackets. These denote a NET/ROM configuration. This configuration is simpler, it simply says that anyone connecting to our NET/ROM port called '`netrom`' will have the *node* program run for them, unless they have a callsign of '`NOCALL`' in which case they will be locked out.

The last two configurations are for incoming ROSE connections. The first for people who have placed calls to '`vk2ktj-0`' and the second for '`VK2KTJ-1` at the our ROSE node address. These work precisely the same way. Not the use of the curly braces to distinguish the port as a ROSE port.

This example is a contrived one but I think it illustrates clearly the important features of the syntax of the configuration file. The configuration file is explained fully in the `ax25d.conf` *man* page. A more detailed example is included in the `ax25-utils` package that might be useful to you too.

12.3. Starting *ax25d*

When you have the two configuration files completed you start *ax25d* with the command:

```
# /usr/sbin/ax25d
```

When this is run people should be able to make AX.25 connections to your Linux machine. Remember to put the `ax25d` command in your *rc* files so that it is started automatically when you reboot each time.

13. Configuring the *node* software

The *node* software was developed by Tomi Manninen (mailto:tomi.manninen@hut.fi) and was based on the original PMS program. It provides a fairly complete and flexible node capability that is easily configured. It allows users once they are connected to make Telnet, NET/ROM, ROSE, and AX.25 connections out and to obtain various sorts of information such as Finger, Nodes and Heard lists etc. You can configure the node to execute any Linux command you wish fairly simply.

The node would normally be invoked from the *ax25d* program although it is also capable of being invoked from the TCP/IP *inetd* program to allow users to telnet to your machine and obtain access to it, or by running it from the command line.

13.1. Creating the `/etc/ax25/node.conf` file

The `node.conf` file is where the main configuration of the node takes place. It is a simple text file and its format is as follows:

```
# /etc/ax25/node.conf
# configuration file for the node(8) program.
#
# Lines beginning with '#' are comments and are ignored.

# Hostname
# Specifies the hostname of the node machine
hostname radio.gw.vk2ktj.ampr.org

# Local Network
# allows you to specify what is consider 'local' for the
# purposes of permission checking using nodes.perms.
localnet 44.136.8.96/29

# Hide Ports
# If specified allows you to make ports invisible to users. The
# listed ports will not be listed by the (P)orts command.
hiddenports rose netrom

# Node Identification.
# this will appear in the node prompt
NodeId  LINUX:VK2KTJ-9

# NET/ROM port
```

```
# This is the name of the NET/ROM port that will be used for
# outgoing NET/ROM connections from the node.
NrPort  netrom

# Node Idle Timeout
# Specifies the idle time for connections to this node in seconds.
idletimout 1800

# Connection Idle Timeout
# Specifies the idle timer for connections made via this node in
# seconds.
conntimeout 1800

# Reconnect
# Specifies whether users should be reconnected to the node
# when their remote connections disconnect, or whether they
# should be disconnected complete.
reconnect on

# Command Aliases
# Provide a way of making complex node commands simple.
alias  CONV "telnet vk1xwt.ampr.org 3600"
alias  BBS "connect radio vk2xsb"

# External Command Aliases
# Provide a means of executing external commands under the node.
# extcmd <cmdname> <flag> <userid> <command>
# Flag == 1 is the only implemented function.
# <command> is formatted as per ax25d.conf
extcmd  PMS 1 root /usr/sbin/pms pms -u %U -o VK2KTJ

# Logging
# Set logging to the system log. 3 is the noisiest, 0 is disabled.
loglevel 3

# The escape character
# 20 = (Control-T)
EscapeChar      20
```

13.2. Creating the `/etc/ax25/node.perms` file

The *node* allows you to assign permissions to users. These permissions allow you to determine which users should be allowed to make use of options such as the (T)elnet, and (C)onnect commands, for example, and which shouldn't. The `node.perms` file is where this information is stored and contains five key fields. For all fields an asterisk '*' character matches anything. This is useful for building default rules.

user

The first field is the callsign or user to which the permissions should apply. Any SSID value is ignored, so you should just place the base callsign here.

method

Each protocol or access method is also given permissions. For example you might allow users who have connected via AX.25 or NET/ROM to use the (C)onnect option, but prevent others, such as those who are telnet connected from a non-local node from having access to it. The second field therefore allows you to select which access method this permissions rule should apply to. The access methods allowed are:

Method	**Description**
ampr	User is telnet connected from an amprnet address (44.0.0.0)
ax25	User connected by AX.25
host	User started node from command line
inet	user is telnet connected from a non-loca, non-ampr address.
local	User is telnet connected from a 'local' host
netrom	User connected by NET/ROM
rose	User connected by ROSE
*	User connected by any means.

port

For AX.25 users you can control permissions on a port by port basis too if you choose. This allows you to determine what AX.25 are allowed to do based on which of your ports they have connected to. The third field contains the port name if you are using this facility. This is useful only for AX.25 connections.

password

You may optionally configure the node so that it prompts users to enter a password when they connect. This might be useful to help protect specially configured users who have high authority levels. If the fourth field is set then its value will be the password that will be accepted.

permissions

The permissions field is the final field in each entry in the file. The permissions field is coded as a bit field, with each facility having a bit value which if set allows the option to be used and if not set prevents the facility being used. The list of controllable facilities and their corresponding bit values are:

Value	**Description**
1	Login allowed.
2	AX.25 (C)onnects allowed.

Value	Description
4	NET/ROM (C)onnects allowed.
8	(T)elnet to local hosts allowed.
16	(T)elnet to amprnet (44.0.0.0) hosts allowed.
32	(T)elnet to non-local, non-amprnet hosts allowed.
64	Hidden ports allowed for AX.25 (C)onnects.
128	ROSE (C)onnects allowed.

To code the permissions value for a rule, simply take each of the permissions you want that user to have and add their values together. The resulting number is what you place in field five.

A sample `nodes.perms` might look like:

```
# /etc/ax25/node.perms
#
# The node operator is VK2KTJ, has a password of 'secret' and
# is allowed all permissions by all connection methods
vk2ktj * * secret 255

# The following users are banned from connecting
NOCALL * * * 0
PK232 * * * 0
PMS * * * 0

# INET users are banned from connecting.
* inet * * 0

# AX.25, NET/ROM, Local, Host and AMPR users may (C)onnect and (T)elnet
# to local and ampr hosts but not to other IP addresses.
* ax25 * * 159
* netrom * * 159
* local * * 159
* host * * 159
* ampr * * 159
```

13.3. Configuring *node* to run from *ax25d*

The *node* program would normally be run by the *ax25d* program. To do this you need to add appropriate rules to the `/etc/ax25/ax25d.conf` file. In my configuration I wanted users to have a choice of either connecting to the *node* or connecting to other services. *ax25d* allows you to do this by cleverly creating port aliases. For example, given the *ax25d* configuration presented above, I want to configure *node* so that all users who connect to `VK2KTJ-1` are given the node. To do this I add the following to my `/etc/ax25/ax25d.conf` file:

```
[vk2ktj-1 via radio]
default    *     *    *   *   *   0    root /usr/sbin/node node
```

This says that the Linux kernel code will answer any connection requests for the callsign '`VK2KTJ-1`' heard on the AX.25 port named '`radio`', and will cause the *node* program to be run.

13.4. Configuring *node* to run from *inetd*

If you want users to be able to telnet a port on your machine and obtain access to the *node* you can go this fairly easily. The first thing to decide is what port users should connect to. In this example I've arbitrarily chosen port 4000, though Tomi gives details on how you could replace the normal telnet daemon with the *node* in his documentation.

You need to modify two files.

To `/etc/services` you should add:

```
node 3694/tcp #OH2BNS's node software
```

and to `/etc/inetd.conf` you should add:

```
node stream tcp nowait root /usr/sbin/node node
```

When this is done, and you have restarted the *inetd* program any user who telnet connects to port 3694 of your machine will be prompted to login and if configured, their password and then they will be

connected to the *node*.

14. Configuring *axspawn*

The *axspawn* program is a simple program that allows AX.25 stations who connect to be logged in to your machine. It may be invoked from the *ax25d* program as described above in a manner similar to the *node* program. To allow a user to log in to your machine you should add a line similar to the following into your `/etc/ax25/ax25d.conf` file:

```
default * * * * * 1 root /usr/sbin/axspawn axspawn %u
```

If the line ends in the + character then the connecting user must hit return before they will be allowed to login. The default is to not wait. Any individual host configurations that follow this line will have the *axspawn* program run when they connect. When *axspawn* is run it first checks that the command line argument it is supplied is a legal callsign, strips the SSID, then it checks that `/etc/passwd` file to see if that user has an account configured. If there is an account, and the password is either `""` (null) or + then the user is logged in, if there is anything in the password field the user is prompted to enter a password. If there is not an existing account in the `/etc/passwd` file then *axspawn* may be configured to automatically create one.

14.1. Creating the `/etc/ax25/axspawn.conf` file

You can alter the behaviour of *axspawn* in various ways by use of the `/etc/ax25/axspawn.conf` file. This file is formatted as follows:

```
# /etc/ax25/axspawn.conf
#
# allow automatic creation of user accounts
create    yes
#
# guest user if above is 'no' or everything else fails. Disable with "no"
guest     no
#
# group id or name for autoaccount
group     ax25
#
# first user id to use
first_uid 2001
#
# maximum user id
```

```
max_uid   3000
#
# where to add the home directory for the new users
home      /home/ax25
#
# user shell
shell     /bin/bash
#
# bind user id to callsign for outgoing connects.
associate yes
```

The eight configurable characteristics of *axspawn* are as follows:

#

indicates a comment.

create

if this field is set to `yes` then *axspawn* will attempt to automatically create a user account for any user who connects and does not already have an entry in the `/etc/passwd` file.

guest

this field names the login name of the account that will be used for people who connect who do not already have accounts if *create* is set to `no`. This is usually `ax25` or `guest`.

group

this field names the group name that will be used for any users who connect and do not already have an entry in the `/etc/passwd` file.

first_uid

this is the number of the first userid that will be automatically created for new users.

max_uid

this is the maximum number that will be used for the userid of new users.

home

this is the home (login) directory of new users.

shell

this is the login shell of any new users.

associate

this flag indicates whether outgoing AX.25 connections made by this user after they login will use their own callsign, or your stations callsign.

15. Configuring the *pms*

The *pms* program is an implementation of a simple personal message system. It was originally written by Alan Cox. Dave Brown (mailto:dcb@vectorbd.com), N2RJT, has taken on further development of it. At present it is still very simple, supporting only the ability to send mail to the owner of the system and to obtain some limited system information but Dave is working to expand its capability to make it more useful.

After that is done there are a couple of simple files that you should create that give users some information about the system and then you need to add appropriate entries into the `ax25d.conf` file so that connected users are presented with the PMS.

15.1. Create the `/etc/ax25/pms.motd` file

The `/etc/ax25/pms.motd` file contains the 'message of the day' that users will be presented with after they connect and receive the usual BBS id header. The file is a simple text file, any text you include in this file will be sent to users.

15.2. Create the `/etc/ax25/pms.info` file

The `/etc/ax25/pms.info` file is also a simple text file in which you would put more detailed information about your station or configuration. This file is presented to users in response to their issuing of the `Info` command from the `PMS>` prompt.

15.3. Associate AX.25 callsigns with system users

When a connected user sends mail to an AX.25 callsign, the *pms* expects that callsign to be mapped, or associated with a real system user on your machine. This is described in a section of its own.

15.4. Add the PMS to the `/etc/ax25/ax25d.conf` file

Adding the *pms* to your `ax25d.conf` file is very simple. There is one small thing you need to think about though. Dave has added command line arguments to the PMS to allow it to handle a number of different text end-of-line conventions. AX.25 and NET/ROM by convention expect the end-of-line to be *carriage return, linefeed* while the standard UNIX end-of-line is just *newline*. So, for example, if you

wanted to add an entry that meant that the default action for a connection received on an AX.25 port is to start the PMS then you would add a line that looked something like:

```
default  1  10 5 100 5   0    root  /usr/sbin/pms pms -a -o vk2ktj
```

This simply runs the *pms* program, telling it that it is an AX.25 connection it is connected to and that the PMS owner is `vk2ktj`. Check the *man* page for what you should specify for other connection methods.

15.5. Test the PMS

To test the PMS, you can try the following command from the command line: # /usr/sbin/pms -u vk2ktj -o vk2ktj Substitute your own callsign for mine and this will run the pms, telling it that it is to use the UNIX end-of-line convention, and that user logging in is `vk2ktj`. You can do all the things connected users can.

Additionally you might try getting some other node to connect to you to confirm that your `ax25d.conf` configuration works.

16. Configuring the *user_call* programs

The '*user_call*' programs are really called: *ax25_call* and *netrom_call*. They are very simple programs designed to be called from *ax25d* to automate network connections to remote hosts. They may of course be called from a number of other places such as shell scripts or other daemons such as the *node* program.

They are like a very simple *call* program. They don't do any meddling with the data at all, so the end of line handling you'll have to worry about yourself.

Let's start with an example of how you might use them. Imagine you have a small network at home and that you have one linux machine acting as your Linux radio gateway and another machine, lets say a BPQ node connected to it via an ethernet connection.

Normally if you wanted radio users to be able to connect to the BPQ node they would either have to digipeat through your linux node, or connect to the node program on your linux node and then connect from it. The *ax25_call* program can simplify this if it is called from the *ax25d* program.

Imagine the BPQ node has the callsign `VK2KTJ-9` and that the linux machine has the AX.25/ethernet port named '`bpq`'. Let us also imagine the Linux gateway machine has a radio port called '`radio`'.

An entry in the `/etc/ax25/ax25d.conf` that looked like:

```
[VK2KTJ-1 via radio]
default    * * * *   *   *  *
  root /usr/sbin/ax25_call ax25_call bpq %u vk2ktj-9
```

would enable users to connect direct to '`VK2KTJ-1`' which would actually be the Linux *ax25d* daemon and then be automatically switched to an AX.25 connection to '`VK2KTJ-9`' via the '`bpq`' interface.

There are all sorts of other possible configurations that you might try. The '*netrom_call*' and '*rose_call*' utilities work in similar ways. One amateur has used this utility to make connections to a remote BBS easier. Normally the users would have to manually enter a long connection string to make the call so he created an entry that made the BBS appear as though it were on the local network by having his *ax25d* proxy the connection to the remote machine.

17. Configuring the ROSE Uplink and Downlink commands

If you are familiar with the ROM based ROSE implementation you will be familiar with the method by which AX.25 users make calls across a ROSE network. If a users local ROSE node has the callsign `VK2KTJ-5` and the AX.25 user wants to connect to `VK5XXX` at remote ROSE node `5050882960` then they would issue the command:

```
c vk5xxx v vk2ktj-5 5050 882960
```

At the remote node, `VK5XXX` would see an incoming connection with the local AX.25 users callsign and being digipeated via the remote ROSE nodes callsign.

The Linux ROSE implementation does not support this capability in the kernel, but there are two application programs called *rsuplnk* and *rsdwnlnk* which perform this function.

17.1. Configuring a ROSE downlink

To configure your Linux machine to accept a ROSE connection and establish an AX.25 connection to any destination callsign that is not being listened for on your machine you need to add an entry to your

`/etc/ax25/ax25d.conf` file. Normally you would configure this entry to be the default behaviour for incoming ROSE connections. For example you might have ROSE listeners operating for destinations like `NODE-0` or `HEARD-0` that you wish to handle locally, but for all other destination calls you may want to pass them to the *rsdwnlink* command and assume they are AX.25 users.

A typical configuration would look like:

```
#
{* via rose}
NOCALL   * * * * * *  L
default  * * * * * *  - root  /usr/sbin/rsdwnlnk rsdwnlnk 4800 vk2ktj-5
#
```

With this configuration any user who established a ROSE connection to your Linux nodes address with a destination call of something that you were not specifically listening for would be converted into an AX.25 connection on the AX.25 port named `4800` with a digipeater path of `VK2KTJ-5`.

17.2. Configuring a ROSE uplink

To configure your Linux machine to accept AX.25 connections in the same way that a ROM ROSE node would you must add an entry into your `/etc/ax25/ax25d.conf` file that looks similar to the following:

```
#
[VK2KTJ-5* via 4800]
NOCALL   * * * * * *  L
default  * * * * * *  - root  /usr/sbin/rsuplnk rsuplnk rose
#
```

Note the special syntax for the local callsign. The '`*`' character indicates that the application should be invoked if the callsign is heard in the digipeater path of a connection.

This configuration would allow an AX.25 user to establish ROSE calls using the example connect sequence presented in the introduction. Anybody attempting to digipeat via `VK2KTJ-5` on the AX.25 port named `4800` would be handled by the *rsuplnk* command.

18. Associating AX.25 callsigns with Linux users

There are a number of situations where it is highly desirable to associate a callsign with a linux user account. One example might be where a number of amateur radio operators share the same linux machine and wish to use their own callsign when making calls. Another is the case of PMS users wanting to talk to a particular user on your machine.

The AX.25 software provides a means of managing this association of linux user account names with callsigns. We've mentioned it once already in the PMS section, but I'm spelling it out here to be sure you don't miss it.

You make the association with the *axparms* command. An example looks like:

```
# axparms -assoc vk2ktj terry
```

This command associates that AX.25 callsign `vk2ktj` with the user `terry` on the machine. So, for example, any mail for `vk2ktj` on the *pms* will be sent to Linux account `terry`.

Remember to put these associations into your *rc* file so that they are available each time your reboot.

Note you should never associate a callsign with the `root` account as this can cause configuration problems in other programs.

19. Configuring APRS

Note: This section has yet to be written. It should cover aprsd, aprsdigi, aprsmon, xastir, JavAPRS, etc.

20. The `/proc/` file system entries

The `/proc` filesystem contains a number of files specifically related to the AX.25 and NET/ROM kernel software. These files are normally used by the AX52 utilities, but they are plainly formatted so you may be interested in reading them. The format is fairly easily understood so I don't think much explanation will be necessary.

/proc/net/arp

contains the list of Address Resolution Protocol mappings of IP addresses to MAC layer protocol addresses. These can be AX.25, ethernet or some other MAC layer protocol.

/proc/net/ax25

contains a list of AX.25 sockets opened. These might be listening for a connection, or active sessions.

/proc/net/ax25_bpqether

contains the AX.25 over ethernet BPQ style callsign mappings.

/proc/net/ax25_calls

contains the linux userid to callsign mappings set my the *axparms -assoc* command.

/proc/net/ax25_route

contains AX.25 digipeater path information.

/proc/net/nr

contains a list of NET/ROM sockets opened. These might be listening for a connection, or active sessions.

/proc/net/nr_neigh

contains information about the NET/ROM neighbours known to the NET/ROM software.

/proc/net/nr_nodes

contains information about the NET/ROM nodes known to the NET/ROM software.

/proc/net/rose

contains a list of ROSE sockets opened. These might be listening for a connection, or active sessions.

/proc/net/rose_nodes

contains a mapping of ROSE destinations to ROSE neighbours.

/proc/net/rose_neigh

contains a list of known ROSE neighbours.

/proc/net/rose_routes

contains a list of all established ROSE connections.

21. AX.25, NET/ROM, ROSE network programming

Probably the biggest advantage of using the kernel based implementations of the amateur packet radio protocols is the ease with which you can develop applications and programs to use them.

While the subject of Unix Network Programming is outside the scope of this document I will describe the elementary details of how you can make use of the AX.25, NET/ROM and ROSE protocols within your software.

21.1. The address families

Network programming for AX.25, NET/ROM and ROSE is quite similar to programming for TCP/IP under Linux. The major differences being the address families used, and the address structures that need to be mangled into place.

The address family names for AX.25, NET/ROM and ROSE are `AF_AX25`, `AF_NETROM` and `AF_ROSE` respectively.

21.2. The header files

You must always include the '`netax25/ax25.h`' header file, and also the '`netrom/netrom.h`' or '`netrose/rose.h`' header files if you are dealing with those protocols. Simple top level skeletons would look something like the following:

For AX.25:

```
#include <netax25/ax25.h>
int s, addrlen = sizeof(struct full_sockaddr_ax25);
struct full_sockaddr_ax25 sockaddr;
sockaddr.fsa_ax25.sax25_family = AF_AX25
```

For NET/ROM:

```
#include <netax25/ax25.h>
#include <netrom/netrom.h>
int s, addrlen = sizeof(struct full_sockaddr_ax25);
struct full_sockaddr_ax25 sockaddr;
sockaddr.fsa_ax25.sax25_family = AF_NETROM;
```

For ROSE:

```
#include <netax25/ax25.h>
#include <netrose/rose.h>
int s, addrlen = sizeof(struct sockaddr_rose);
struct sockaddr_rose sockaddr;
sockaddr.srose_family = AF_ROSE;
```

21.3. Callsign mangling and examples

There are routines within the `lib/ax25.a` library built in the AX.25 utilities package that manage the callsign conversions for you. You can write your own of course if you wish.

The *user_call* utilities are excellent examples from which to work. The source code for them is included in the AX.25 utilities package. If you spend a little time working with those you will soon see that ninety percent of the work is involved in just getting ready to open the socket. Actually making the connection is easy, the preparation takes time.

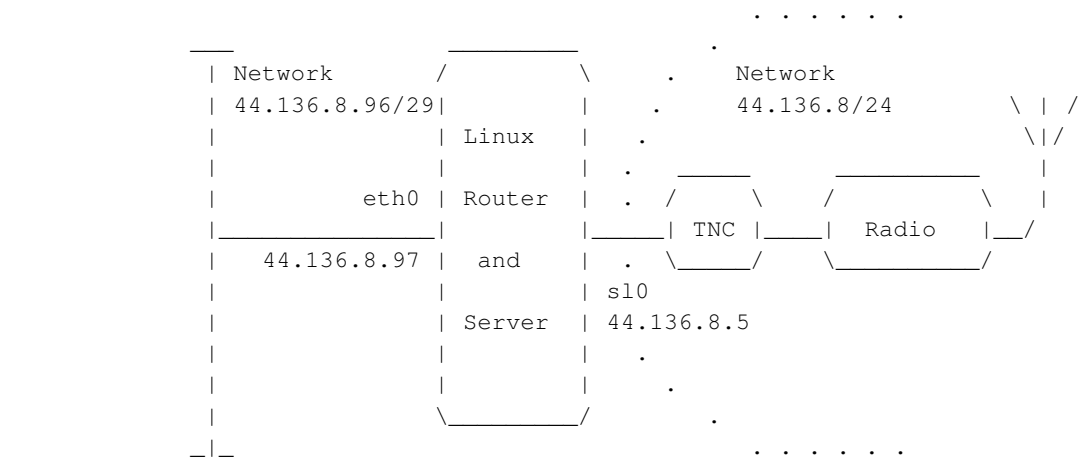
The examples are simple enough to not be very confusing. If you have any questions, you should feel to direct them to the `linux-hams` mailing list and someone there will be sure to help you.

22. Some sample configurations

Following are examples of the most common types of configurations. These are guides only as there are as many ways of configuring your network as there are networks to configure, but they may give you a start.

22.1. Small Ethernet LAN with Linux as a router to Radio LAN

Many of you may have small local area networks at home and want to connect the machines on that network to your local radio LAN. This is the type of configuration I use at home. I arranged to have a suitable block of addresses allocated to me that I could capture in a single route for convenience and I use these on my Ethernet LAN. Your local IP coordinator will assist you in doing this if you want to try it as well. The addresses for the Ethernet LAN form a subset of the radio LAN addresses. The following configuration is the actual one for my linux router on my network at home:




```
#!/bin/sh
# /etc/rc.net
# This configuration provides one KISS based AX.25 port and one
# Ethernet device.

echo "/etc/rc.net"
echo "  Configuring:"

echo -n "    loopback:"
/sbin/ifconfig lo 127.0.0.1
/sbin/route add 127.0.0.1
echo " done."

echo -n "    ethernet:"
/sbin/ifconfig eth0 44.136.8.97 netmask 255.255.255.248 \
  broadcast 44.136.8.103 up
/sbin/route add 44.136.8.97 eth0
/sbin/route add -net 44.136.8.96 netmask 255.255.255.248 eth0
echo " done."

echo -n "    AX.25: "
kissattach -i 44.136.8.5 -m 512 /dev/ttyS1 4800
ifconfig sl0 netmask 255.255.255.0 broadcast 44.136.8.255
route add -host 44.136.8.5 sl0
route add -net 44.136.8.0 window 1024 sl0

echo -n "    NET/ROM: "
nrattach -i 44.136.8.5 netrom

echo "  Routing:"
/sbin/route add default gw 44.136.8.68 window 1024 sl0
echo "    default route."
echo done.
```

```
# end
```

```
/etc/ax25/axports

# name  callsign        speed   paclen  window  description
4800    VK2KTJ-0        4800    256     2       144.800 MHz
```

```
/etc/ax25/nrports

# name  callsign        alias   paclen  description
netrom  VK2KTJ-9        LINUX   235     Linux Switch Port
```

```
/etc/ax25/nrbroadcast

# ax25_name     min_obs def_qual        worst_qual      verbose
4800            1       120             10              1
```

Note the following:

- You must have IP_FORWARDING enabled in your kernel.
- The AX.25 configuration files are pretty much those used as examples in the earlier sections, refer to those where necessary.
- I've chosen to use an IP address for my radio port that is not within my home network block. I needn't have done so, I could have easily used `44.136.8.97` for that port too.
- `44.136.8.68` is my local IPIP encapsulated gateway and hence is where I point my default route.
- Each of the machines on my Ethernet network have a route:

  ```
  route add -net 44.0.0.0 netmask 255.0.0.0 \
   gw 44.136.8.97 window 512 mss 512 eth0
  ```

  The use of the *mss* and *window* parameters means that I can get optimum performance from both local Ethernet and radio based connections.

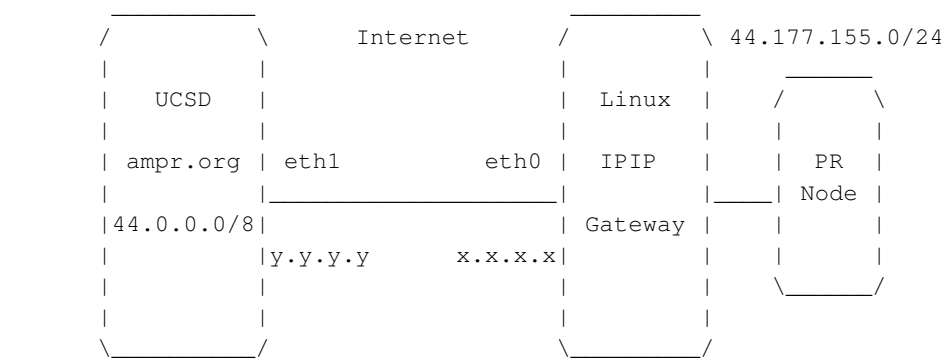
- I also run my smail, http, ftp and other daemons on the router machine so that it needs to be the only machine to provide others with facilities.

- The router machine is a lowly 386DX20 with a 20Mb hard drive and a very minimal linux configuration.

22.2. IPIP encapsulated gateway configuration

Linux is now very commonly used for TCP/IP encapsulated gateways around the world. The 2.2 and 2.4 kernels provide a new method, making the old `ipip` configuration obsolete. The `ip` command contained in the IPROUTE2 package is now the main tool, as described in the Linux 2.4 Advanced Routing HOWTO (http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html).

A typical configuration would look similar to the following:



The configuration file for this example is the following:

```
# /etc/rc.d/rc.tunnel
# This file is a simple configuration that provides the IPIP encapsulation,
# commonly used when utilising the ampr.org (44.0.0.0/8) routing via UCSD.
# The script is located on IPIP gateway with eth0 interface, connected directly
# to the internet and other (e.g. sl0) interface, connected to packet radio
# subnet, e.g. 44.177.155/24.
#
IP_eth0=x.x.x.x
IP_eth1=y.y.y.y
echo "  Configuring:"
#
ip tunnel add ucsd remote $IP_eth1 mode ipip
# 'ucsd' is (any suitable) tunnel name
ifconfig ucsd $IP_eth0 up
# tunnel initialisation
```

```
ip route add 44/8 dev ucsd via $IP_eth1 onlink
# tells that tunnel should be used when sending packets to ampr.org network
# onlink is the magic word, do not forget
echo " done."
#
# end.
```

In any case, the tunnel must be set up on both sides of the route. The tunnelling interface configured above is used for both encapsulation and decapsulation. However, the same principle can be used for one of those tasks, exclusively. When needed, the standard routing (via UCSD), used in previous example, can be avoided by setting the IPIP tunneling between two PR stations, where only one of them has its own internet (public) non-ampr IP address. The task is then to set up the one-way IPIP tunnel, to achieve a quicker and more stable route from non-ampr IP address to ampr IP address station. In this case, the setup, mentioned above, is used for encapsulation. The other side of the route can leave out the route setting, due to its pure decapsulation task.

22.3. AXIP encapsulated gateway configuration

Many Amateur Radio Internet gateways encapsulate AX.25, NET/ROM and ROSE in addition to tcp/ip. Encapsulation of AX.25 frames within IP datagrams is described in RFC-1226 by Brian Kantor. Mike Westerhof wrote an implementation of an AX.25 encapsulation daemon for UNIX in 1991. The ax25-utils package includes a marginally enhanced version of it for Linux.

An AXIP encapsulation program accepts AX.25 frames at one end, looks at the destination AX.25 address to determine what IP address to send them to, encapsulates them in a tcp/ip datagram and then transmits them to the appropriate remote destination. It also accepts tcp/ip datagrams that contain AX.25 frames, unwraps them and processes them as if it had received them directly from an AX.25 port. To distinguish IP datagrams containing AX.25 frames from other IP datagrams which don't, AXIP datagrams are coded with a protocol id of 4 (or 94 which is now deprecated). This process is described in RFC-1226.

The *ax25ipd* program included in the ax25-utils package presents itself as a program supporting a KISS interface across which you pass AX.25 frames, and an interface into the tcp/ip protocols. It is configured with a single configuration file called `/etc/ax25/ax25ipd.conf`.

22.3.1. AXIP configuration options

The *ax25ipd* program has two major modes of operation. "digipeater" mode and "tnc" mode. In "tnc" mode the daemon is treated as though it were a kiss TNC, you pass KISS encapsulated frames to it and it will transmit them, this is the usual configuration. In "digipeater" mode, you treat the daemon as though it were an AX.25 digipeater. There are subtle differences between these modes.

In the configuration file you configure "routes" or mappings between destination AX.25 callsigns and the IP addresses of the hosts that you want to send the AX.25 packets too. Each route has options which will be explained later.

Other options that are configured here are:

- the tty that the *ax25ipd* daemon will open and its speed (usually one end of a pipe)
- what callsign you want to use in "digipeater" mode
- beacon interval and text
- whether you want to encapsulate the AX.25 frames in IP datagrams or in UDP/IP datagrams. Nearly all AXIP gateways use IP encapsulation, but some gateways are behind firewalls that will not allow IP with the AXIP protocol id to pass and are forced to use UDP/IP. Whatever you choose must match what the tcp/ip host at the other end of the link is using.

22.3.2. A typical `/etc/ax25/ax25ipd.conf` file

```
#
# ax25ipd configuration file for station floyd.vk5xxx.ampr.org
#
# Select axip transport. 'ip' is what you want for compatibility
# with most other gateways.
#
socket ip
#
# Set ax25ipd mode of operation. (digi or tnc)
#
mode tnc
#
# If you selected digi, you must define a callsign. If you selected
# tnc mode, the callsign is currently optional, but this may change
# in the future! (2 calls if using dual port kiss)
#
#mycall vk5xxx-4
#mycall2 vk5xxx-5
#
# In digi mode, you may use an alias. (2 for dual port)
#
#myalias svwdns
#myalias2 svwdn2
#
# Send an ident every 540 seconds ...
#
#beacon after 540
#btext ax25ip -- tncmode rob/vk5xxx -- Experimental AXIP gateway
#
# Serial port, or pipe connected to a kissattach in my case
```

```
#
device /dev/ttyq0
#
# Set the device speed
#
speed 9600
#
# loglevel 0 - no output
# loglevel 1 - config info only
# loglevel 2 - major events and errors
# loglevel 3 - major events, errors, and AX.25 frame trace
# loglevel 4 - all events
# log 0 for the moment, syslog not working yet ...
#
loglevel 2
#
# If we are in digi mode, we might have a real tnc here, so use param to
# set the tnc parameters ...
#
#param 1 20
#
# Broadcast Address definition. Any of the addresses listed will be forwarded
# to any of the routes flagged as broadcast capable routes.
#
broadcast QST-0 NODES-0
#
# ax.25 route definition, define as many as you need.
# format is route (call/wildcard) (ip host at destination)
# ssid of 0 routes all ssid's
#
# route <destcall> <destaddr> [flags]
#
# Valid flags are:
#         b  - allow broadcasts to be transmitted via this route
#         d  - this route is the default route
#
route vk2sut-0 44.136.8.68 b
route vk5xxx 44.136.188.221 b
route vk2abc 44.1.1.1
#
#
```

22.3.3. Running *ax25ipd*

Create your `/etc/ax25/axports` entry:

```
# /etc/ax25/axports
```

```
#
axip VK2KTJ-13 9600 256 AXIP port
#
```

Run the *kissattach* command to create that port:

```
/usr/sbin/kissattach /dev/ptyq0 axip 44.135.96.242
```

Run the *ax25ipd* program:

```
/usr/sbin/ax25ipd &
```

Test the AXIP link:

```
call axip vk5xxx
```

22.3.4. Some notes about the routes and route flags

The "`route`" command is where you specify where you want your AX.25 packets encapsulated and sent to. When the *ax25ipd* daemon receives a packet from its interface, it compares the destination callsign with each of the callsigns in its routing table. If it finds a match then the ax.25 packet is encapsulated in an IP datagram and then transmitted to the host at the specified IP address.

There are two flags you can add to any of the route commands in the `ax25ipd.conf` file. The two flags are:

b

: traffic with a destination address matching any of those on the list defined by the "`broadcast`" keyword should be transmitted via this route.

d

: any packets not matching any route should be transmitted via this route.

The broadcast flag is very useful, as it enables informations that is normally destined for all stations to a number of AXIP destinations. Normally axip routes are point-to-point and unable to handle 'broadcast' packets.

22.4. Linking NOS and Linux using a pipe device

Many people like to run some version of NOS under Linux because it has all of the features and facilities they are used to. Most of those people would also like to have the NOS running on their machine capable of talking to the Linux kernel so that they can offer some of the linux capabilities to radio users via NOS.

Brandon S. Allbery, KF8NH, contributed the following information to explain how to interconnect the NOS running on a Linux machine with the kernel code using the Linux pipe device.

Since both Linux and NOS support the slip protocol it is possible to link the two together by creating a slip link. You could do this by using two serial ports with a loopback cable between them, but this would be slow and costly. Linux provides a feature that many other Unix-like operating systems provide called 'pipes'. These are special pseudo devices that look like a standard tty device to software but in fact loopback to another pipe device. To use these pipes the first program must open the *master* end of the pipe, and the open then the second program can open the *slave* end of the pipe. When both ends are open the programs can communicate with each other simply by writing characters to the pipes in the way they would if they were terminal devices.

To use this feature to connect the Linux Kernel and a copy of NOS, or some other program you first must choose a pipe device to use. You can find one by looking in your `/dev` directory. The master end of the pipes are named: `ptyq[1-f]` and the slave end of the pipes are known as: `ttyq[1-f]`. Remember they come in pairs, so if you select `/dev/ptyqf` as your master end then you must use `/dev/ttyqf` as the slave end.

Once you have chosen a pipe device pair to use you should allocate the master end to you linux kernel and the slave end to the NOS program, as the Linux kernel starts first and the master end of the pipe must be opened first. You must also remember that your Linux kernel must have a different IP address to your NOS, so you will need to allocate a unique address for it if you haven't already.

You configure the pipe just as if it were a serial device, so to create the slip link from your linux kernel you can use commands similar to the following:

```
# /sbin/slattach -s 38400 -p slip /dev/ptyqf &
# /sbin/ifconfig sl0 broadcast 44.255.255.255 pointopoint 44.70.248.67 /
 mtu 1536 44.70.4.88
# /sbin/route add 44.70.248.67 sl0
# /sbin/route add -net 44.0.0.0 netmask 255.0.0.0 gw 44.70.248.67
```

In this example the Linux kernel has been given IP address `44.70.4.88` and the NOS program is using IP address `44.70.248.67`. The *route* command in the last line simply tells your linux kernel to route all datagrams for the amprnet via the slip link created by the *slattach* command. Normally you would put

these commands into your `/etc/rc.d/rc.inet2` file after all your other network configuration is complete so that the slip link is created automatically when you reboot. Note: there is no advantage in using *cslip* instead of *slip* as it actually reduces performance because the link is only a virtual one and occurs fast enough that having to compress the headers first takes longer than transmitting the uncompressed datagram.

To configure the NOS end of the link you could try the following:

```
# you can call the interface anything you want; I use "linux" for convenience.
attach asy ttyqf - slip linux 1024 1024 38400
route addprivate 44.70.4.88 linux
```

These commands will create a slip port named 'linux' via the slave end of the pipe device pair to your linux kernel, and a route to it to make it work. When you have started NOS you should be able to ping and telnet to your NOS from your Linux machine and vice versa. If not, double check that you have made no mistakes especially that you have the addresses configured properly and have the pipe devices around the right way.

23. Summary of AX.25-related Linux commands

This section summarizes all of the commands that are specific to AX.25.

Command	Package	Description
mheard	ax25-tools	Display AX.25 calls recently heard
ax25d	ax25-tools	General purpose AX.25, NET/ROM and ROSE daemon
axctl	ax25-tools	Configure/Kill running AX.25 connections
axparms	ax25-tools	Configure AX.25 interfaces
axspawn	ax25-tools	Allow automatic login to a Linux system
beacon	ax25-tools	Transmit periodic messages on an AX.25 port
bpqparms	ax25-tools	Configure BPQ ethernet devices
mheardd	ax25-tools	Collect information about packet activity

Command	Package	Description
rxecho	ax25-tools	Route AX.25 packets between ports transparently
sethdlc	ax25-tools	Get/set Linux HDLC packet radio modem driver port information
smmixer	ax25-tools	Get/set Linux soundcard packet radio modem driver mixer
smdiag	ax25-tools	Linux soundcard packet radio modem driver diagnostics utility
kissattach	ax25-tools	Attach a KISS or 6PACK interface
kissnetd	ax25-tools	Create a virtual network
kissparms	ax25-tools	Configure KISS TNCs
net2kiss	ax25-tools	Convert a network AX.25 driver to a KISS stream on a pseudo-tty
mkiss	ax25-tools	Attach a multi KISS interface
nodesave	ax25-tools	Saves NET/ROM routing information
nrattach	ax25-tools	Start a NET/ROM interface
nrparms	ax25-tools	Configure the NET/ROM interface
nrsdrv	ax25-tools	KISS to NET/ROM serial converter
netromd	ax25-tools	Send and receive NET/ROM routing messages
rsattach	ax25-tools	Start a ROSE interface
rsdwnlnk	ax25-tools	User exit from the ROSE network
rsparms	ax25-tools	Configure the ROSE interface
rsuplnk	ax25-tools	User entry into the ROSE network
ttylinkd	ax25-tools	TTYlink daemon for AX.25, NET/ROM, ROSE and IP
rip98d	ax25-tools	Send and receive RIP98 routing messages
ax25_call	ax25-tools	Make an AX.25, NET/ROM, ROSE or TCP connection
netrom_call	ax25-tools	Make an AX.25, NET/ROM, ROSE or TCP connection
rose_call	ax25-tools	Make an AX.25, NET/ROM, ROSE or TCP connection

Command	Package	Description
tcp_call	ax25-tools	Make an AX.25, NET/ROM, ROSE or TCP connection
yamcfg	ax25-tools	Configure YAM driver parameters
dmascc_cfg	ax25-tools	Configure dmascc devices
ax25ipd	ax25-apps	AX.25 into IP Encapsulator
ax25rtd	ax25-apps	AX.25 routing daemon
ax25rtctl	ax25-apps	AX.25 routing daemon control utility
call	ax25-apps	Make an AX.25, NET/ROM or ROSE connection
listen	ax25-apps	Monitor AX.25 traffic
ax25mond	ax25-apps	Dump the AX.25 network traffic and provide sockets where the received data will be retransmitted
soundmodem	soundmodem	Soundcard modem driver
soundmodemconfig	soundmodem	Soundcard modem configuration utility
aprsd	aprsd	APRS daemon
aprspass	aprsd	APRS passcode generator
aprsdigi	aprsdigi	APRS digipeater
aprsmon	aprsdigi	Monitor APRS AX.25 traffic for JavAPRS

24. Where do I find more information about .... ?

Since this document assumes you already have some experience with packet radio, and that this might not be the case, I've collected a set of references to other information that you might find useful.

24.1. Packet Radio

You can get general information about Packet Radio from these sites:

- American Radio Relay League (http://www.arrl.org/)
- Radio Amateur Teleprinter Society (http://www.rats.org/)
- Tucson Amateur Packet Radio Group (http://www.tapr.org/)

24.2. Protocol Documentation

- AX.25, NET/ROM - Jonathon Naylor has collated a variety of documents that relate to the packet radio protocols themselves. This documentation has been packaged up into ax25-doc-1.0.tar.gz (ftp://ftp.hes.iki.fi/pub/ham/unix.linux/ax25/ax25-doc-1.0.tar.gz)

24.3. Hardware Documentation

- Information on the *PI2 Card* is provided by the Ottawa Packet Radio Group (http://hydra.carleton.ca/).
- Information on *Baycom hardware* is available at the Baycom Web Page (http://www.baycom.de/).

24.4. Linux Ham Radio Software

John Ackermann has a web site with information related to configuring AX.25 on Linux at http://www.febo.com/linux-ax25/index.html.

The Hamsoft Linux Ham Radio Applications and Utilities Database attempts to maintain a complete list of Amateur Radio related applications for Linux. It can be found at http://radio.linux.org.au (http://radio.linux.org.au/).

25. Discussion relating to Amateur Radio and Linux

There are various places that discussion relating to Amateur Radio and Linux take place. They take place in the `comp.os.linux.*` newsgroups, they also take place on the `linux-hams` list on `vger.kernel.org`. Other places where they are held include the `tcp-group` mailing list at `ucsd.edu` (the home of amateur radio TCP/IP discussions), and you might also try the `#linpeople` channel on the `linuxnet` irc network.

To join the Linux *linux-hams* channel on the mail list server, send mail to `majordomo@vger.kernel.org` with the line `subscribe linux-hams` in the message body. The subject line is ignored.

The *linux-hams* mailing list is archived at: http://hes.iki.fi/archive/linux-hams/ and http://web.gnu.walfield.org/mail-archive/linux-hams. Please use the archives when you are first starting, because many common questions are answered there.

To join the `tcp-group` send mail to `listserver@ucsd.edu` with the line `subscribe tcp-group` in the body of the text.

> **Note:** Please remember that the `tcp-group` is primarily for discussion of the use of advanced protocols, of which TCP/IP is one, in Amateur Radio. *Linux specific questions should not ordinarily go there.*

26. Acknowledgements

Terry Dawson was the original author and maintainer of this HOWTO. Jeff Tranter took over as maintainer in 2001 to allow Terry more time to concentrate on AX.25 software development.

The following people have contributed to this document in one way or another, knowingly or unknowingly. In no particular order (as I find them): Jonathon Naylor, Thomas Sailer, Joerg Reuter, Ron Atkinson, Alan Cox, Craig Small, John Tanner, Brandon Allbery, Hans Alblas, Klaus Kudielka, Carl Makin, John Ackermann, Riley Williams, Milan Kalina.

27. Feedback

I rely on you, the reader, to make this HOWTO useful. If you have any suggestions, corrections, or comments, please send them to me, tranter@pobox.com (mailto:tranter@pobox.com), and I will try to incorporate them in the next revision.

If you publish this document on a CD-ROM or in hardcopy form, a complimentary copy would be appreciated; mail me for my postal address. Also consider making a donation to the Linux Documentation Project to help support free documentation for Linux. Contact the LDP at feedback@linuxdoc.org (mailto:feedback@linuxdoc.org) for more information.

28. Distribution Policy